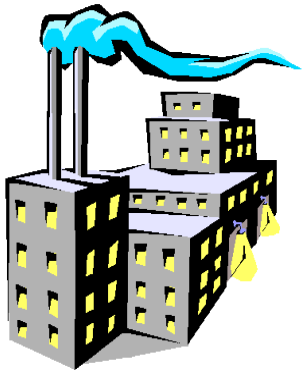
Diplomarbeit

Fernwartungstool für Feldgeräte

Diplomarbeit Nr. MT 06-01.03



Abstract

Feldgeräte (Mikrocontroller), die mit einem Webserver über das HTTP-Protokoll kommunizieren, eröffnen Möglichkeiten zur Fernwartung und Analyse von Automaten oder Maschinen. In dieser Diplomarbeit soll ein System entwickelt werden, welches erlaubt, solche Automaten und Geräte über einen Browser fernzusteuern.

Keywords

Fernwartung, Feldgeräte, Fernwartungstool, Fernsteuerung

Experte
Rolf Lanz
Software-Schule Schweiz
Wankdorffeldstr 102
3014 Bern
031 84 83 273
rolf.lanz@bfh.ch

Betreuer
Hamlet Tamazian
Alpenstrasse 125
3052 Zollikofen
031 911 26 40
hamlet@tamazian.ch

Student
Simon Anderegg
Lehrerhaus Steinenbrünnen
3148 Lanzenhäusern
079 768 32 78
simon@anderegg.org

Inhaltsverzeichnis

1 Allgemeines....4
1.1 Leserkreis....4
1.2 Einführung....5
1.3 Definitionen und Abkürzungen....9
1.4 Abgrenzung des Umfeldes....10
1.5 Feldebene....11
1.5.1 Installation....11
1.5.2 Selbsttest....11
1.5.3 System Setup....11
1.5.4 Systemstart und System-Shutdown....11
1.5.5 Verhalten bei Power Fail....11
1.5.6 Sprachen....11
1.5.7 Hardware....12
1.5.8 Programmierung....13
1.6 Datenverarbeitungsebene....15
1.6.1 Umgebung....15
1.6.2 Programmierung....16
1.6.3 Datenbank....16
1.6.4 Benutzereigenschaften....16
1.7 Systementscheide....16
1.8 Verwendete Software und Tools....17
2 Wahl der Übertragungsart....18
2.1 Wahl des Übertragungsprotokolls....18
2.2 Beteiligte Kommunikationspartner....18
2.3 Datenclients....19
2.4 Varianten....20
2.4.1 Datenaufkommen....21
3 Systemübersicht....23
3.1 Datenfluss Feldgeräte – Datenverarbeitung - Browser....25
4 HTTP-Request zum Server....26
4.1 Design....26
4.2 Umsetzung....27
5 Verarbeitung Feldebene....29
5.1 Design....29
5.2 Umsetzung Schnittstelle zu Feldebene....29
6 Kommunikationssoftware Feldgerät....34
6.1 Design....34
6.2 Umsetzung....41
6.2.1 Serviceschnittstelle....49
7 Datenbank....52
8 Bedienerinterface....53
8.1 Design....53
8.2 Umsetzung....56
9 Kontrollfunktionen und Tests....58

9.1 Allgemeines....58
9.2 Modultests Feldebene....58
10 Tests....61
11 Das HTTP-Protokoll....63
12 Manipulationssicherheit....63
12.1 Allgemeines....63
12.2 Sicherheit Kommunikation Feldebene – Webserver....63
12.3 Sicherheit Kommunikation Webserver - Browser....64
12.4 Sicherheit Datenverarbeitungsebene....64
12.5 Sicherheit Feldebene....65
13 Datenbasis....66
13.1 Hardwareschnittstellen....68
14 Resumée....69
15 Ausblick....70
16 Quellenangaben....71
17 Abbildungsverzeichnis....72

1 Allgemeines

1.1 Leserkreis

Als Feldgeräte, engl. Field Device (FD), werden sowohl Aktoren (Stellglieder, Ventile, ...), wie auch Sensoren (Messumformer) in der Fabrik- und Prozessautomation bezeichnet. Die Feldgeräte sind dann mit einem Steuerungs- und Leitsystem, meist über einen Bus, verbunden. Hier werden die Daten ausgewertet und dienen dann zur Regelung, Steuerung und weiteren Verarbeitung. Dazu gehört z.B. auch die Visualisierung und Anzeige des Ergebnisses (z. B. Ventil geöffnet/geschlossen, Druck, Durchfluss, Temperatur,...) für das Personal.

Das Dokument richtet sich an Privatpersonen oder Firmen, welche abgesetzte elektronische Geräte betreiben respektive einsetzen wollen. Um den Inhalt dieser Arbeit verstehen zu können, sollte der Leser eine Vorstellung von TCP/IP und dem darauf aufbauenden HTTP-Protokoll haben. Grundkenntnisse in den Bereichen Softwareentwicklung und -design sind ebenfalls von Vorteil.

1.2 Einführung

Immer häufiger müssen in Firmen Produktionsdaten erfasst und protokolliert werden. Dies bringt häufig Schnittstellenprobleme und, nicht zu unterschätzen, manchmal auch Konfliktpotential zwischen Automations-und IT–Fachkräften mit sich.

Automations-Fachleute arbeiten häufig in einem anderen Umfeld als ihre Kollegen der IT-Abteilung. Ihr Alltag besteht aus C, AWL(IEC 61131-3), FUPLA, Mikrocontrollern, SPS'en, SCADA undDCS. Die IT-Abteilung hingegen denkt und Programmiert zumeist in Hochsprachen wie Java .NET und SQL. Das Bindeglied zwischen den beiden Welten wird MES (Manufacturing Excecution System) genannt. In der neueren Vergangenheit haben SCADA-Anbieter immer mehr IT-Funktionalitäten wie Prozessdatenerfassung Produktionsstatistiken und Rezeptur-Management in ihre Produkte integriert. Meistens handelt es sich dabei um proprietäre Lösungen. Den Automationsprojekten folgen in vielen Fällen mehr oder weniger umfangreiche Projekte, welche die Kommunikation zwischen den beiden Welten zum Beispiel via OPC (OLE for Process Control) ermöglicht.

OPC wurde von einigen grossen Firmen zwecks Vereinheitlichung des Datenaustausches ins Leben gerufen. Es fusst im wesentlichen auf Microsofts DCOM . Der Server muss demnach ein Microsoft-Betriebssysteme haben. Die Kommunikationen über Firewalls hinweg ist ebenfalls nicht ohne weiteres möglich.

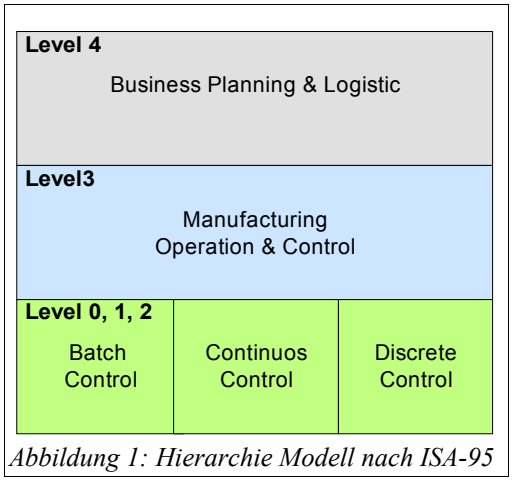


Abbildung 1: Hierarchie Modell nach ISA-95

Aber nicht nur Datenaustausch mit Fremdsystemen ist in zunehmenden Masse gefragt, auch Fernsteuern von Anlagen ist in Zukunft immer mehr ein Thema. So wollen Industriebetriebe und Produktionsbetriebe zentral ihre Produktionsanlagen von einer Leitwarte aus steuern, oder von dezentral verteilten Systemen müssen Status und Fehlermeldungen zentral verwaltet werden können. In den nächsten zwei Beispielen möchte ich die Problematik an zwei Fällen aus der Praxis erläutern.

In meiner bisherigen beruflichen Laufbahn bin ich Datenaustausch-Thematiken des öfteren begegnet. So erstellte ich im Auftrag meines früheren Arbeitgebers Betriebsleitsysteme für Bergbahnen. Diese erfassten unter anderem auch Wetterdaten wie Wind, Luftfeuchtigkeit und Temperatur. Der nahe liegende Gedanke des Kunden war diese Daten für das Gästeinformationssystem zu nutzen. Ein Gästeinformationssystem umfasst die Verteilung von Informationen über verschieden Kanäle wie Leucht-

schriftenanzeigen, Flatscreens, Internet Kabelfernsehen etc. Wie kann die Kommunikation zwischen den beiden Anwendungen schnell und kostengünstig realisiert werden?

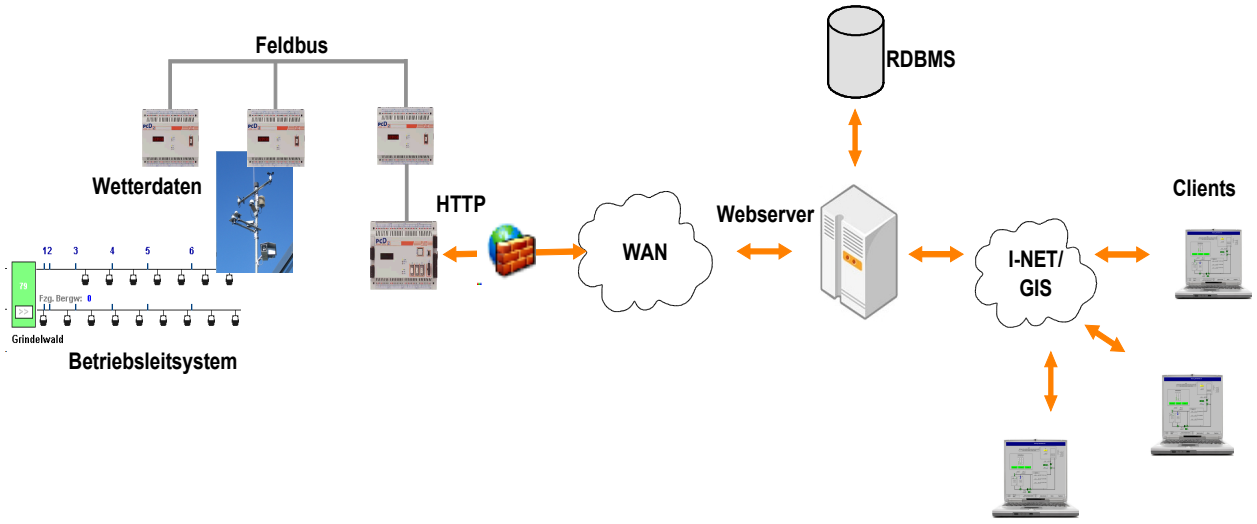


Abbildung 2: Lösungsansatz Kunde 1

Die Abbildung 2 zeigt eine mögliche Lösung des Problems. Eine Kopfstation, ausgerüstet mit einer TCP/IP Schnittstelle sammelt die Daten und sendet sie per HTTP Request durch die Firewall hindurch an die bereits vorhandene Webserver Infrastruktur. Die Daten werden in die bestehende Applikation integriert und stehen den Clients damit zur Verfügung. Die Firewall muss nicht speziell konfiguriert werden, da der HTTP Traffic wie ein „normaler“ Browser Traffic aussieht und daher die Firewall unbehelligt passieren kann.

Kunde 2 stellt elektrische Hochleistungsantriebe für Werkzeugmaschinen her. Für diesen projektierte, und erstellte man die Automationssoftware eines Prüfstandes. Jeder Antrieb durchlief ein vorgegebenes Belastungs Programm und wurde, wenn das Testprozedere erfolgreich verlaufen war, ausgeliefert.

Der Kunde hatte den Wunsch die Protokoll und das Testergebnis zusammen mit der Serienummer des Antriebes auf seinem ERP abzuspeichern. Auch hier stellte sich die Frage: Wie kann der Datenaustausch rasch, unkompliziert, kostengünstig und möglichst ohne zusätzliche Hardware realisiert werden?

Etwas verallgemeinert ist der Wunsch des Kunden 2 in der nächsten Abbildung skizziert. Mehrere Feldbusnetze können wie in der vorderen Anwendung ebenfalls mit einer TCP/IP-Kopfstation ausgerüstet werden. Diese kommunizieren mit einem Webserver über das HTTP Protokoll. Am Webserver angegliedert ist ein RDBMS System. Dieses ist die Schnittstelle zum ERP. Es kann als eigenständiges System betrieben werden, kann aber auch Bestandteil des ERP's sein, oder im Webserver integriert werden.

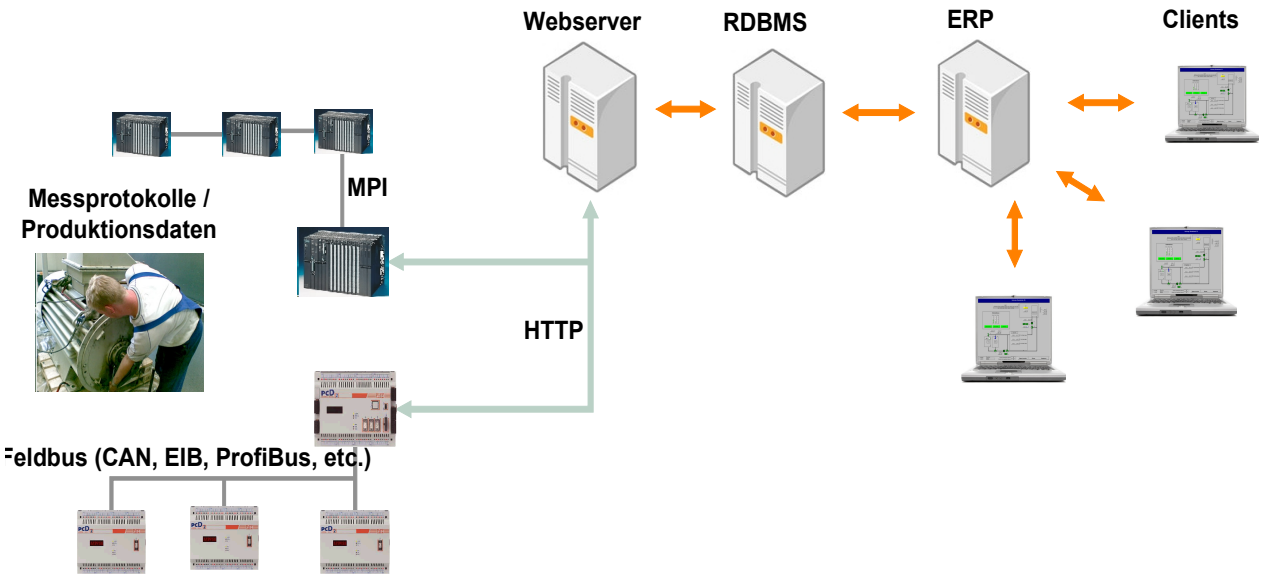


Abbildung 3: Lösungsansatz Kunde 2

In dieser Arbeit möchte ich an einem einem konkreten Beispiel eine Möglichkeit aufzeigen, wie die Automation und die IT-Welt sich weiter annähern könnten:

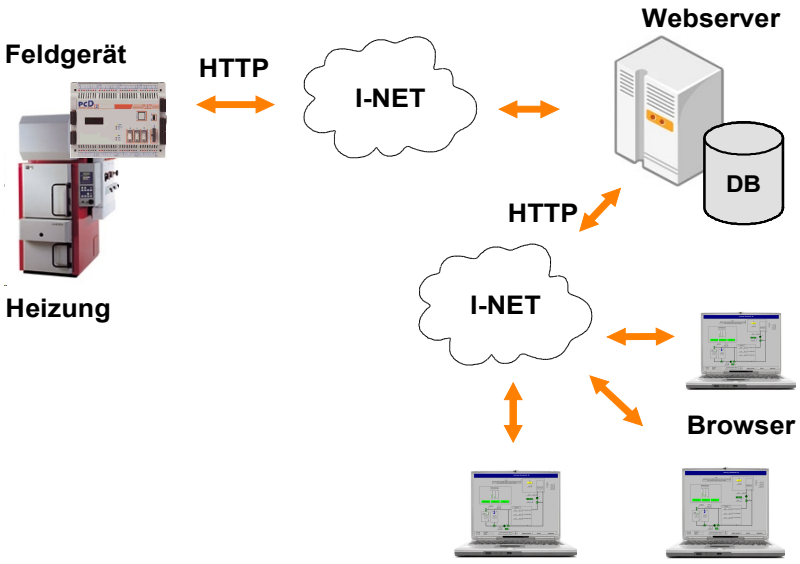


Abbildung 4: Übersicht Beispiel-Implementation

Einer bestehenden Heizungsanlage deren Steuerung mit einer IP-Schnittstelle ausgerüstet ist, sendet in regelmässigen Abständen Ihre Daten per HTTP_Request zum Webserver. Das Prozessabbild wird in der Datenbank gespeichert. Die Prozessdaten können im Webbrowser, der die Daten auf dem Webserver abruft, eingesehen werden.

Zusammengefasst lassen sich die oben genannten Problemstellungen im nachfolgendem Use–Case ausgedrücken:

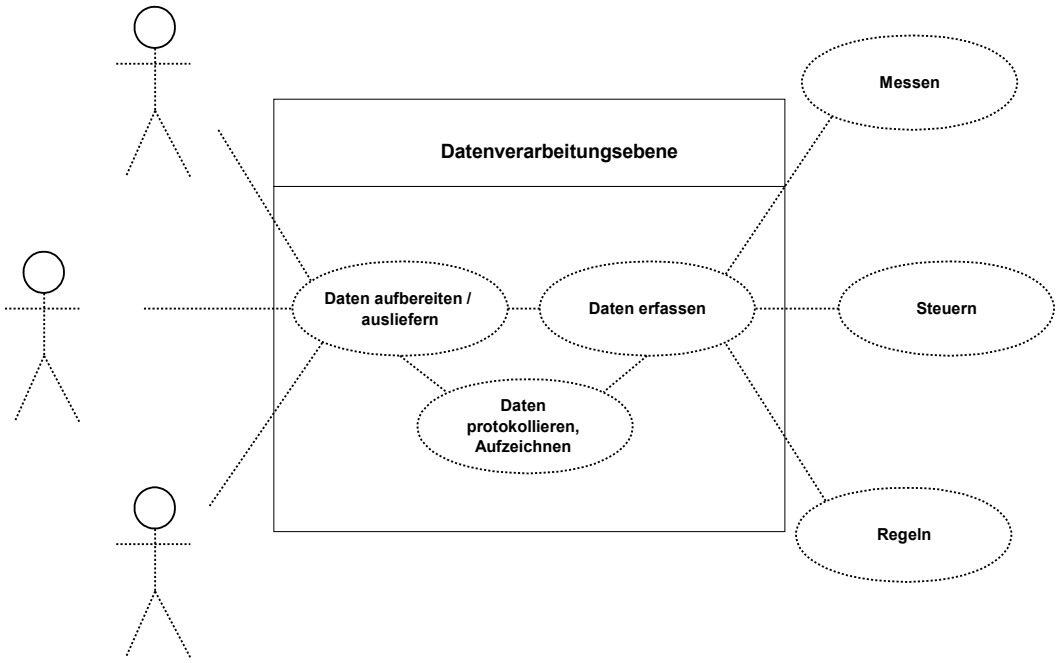


Abbildung 5: Übersicht use-case

Prozessdaten müssen erfasst, protokolliert werden und anderen Systemen oder Benutzern verfügbar gemacht werden.

1.3 Definitionen und Abkürzungen

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
AWL	Anweisungsliste (IEC 61131-3)
CSS	Cascading Style Sheets
CMS	Content Management System
CISC	Complex Instruction Set Computing
CR LF	Carriage Return/Line Feed (Wagen Rücklauf, Zeilenvorschub)
DB	Database (Datenbank)
DCS	Distributed Control Systems
DCOM	Distributed Component Object Model
ERP	Enterprise Resource Planning
FUPLA	Funktionsplan
GUI	Grpahical User Interface
HTML	Hypertext Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IEC	International Electrotechnical Commission
IP	Internet Protocol
IT	Information Technology
ISA	Instrument Society of Amerika
RFC	Request for Comments
RDBMS	Relational database management system
RPC	Remote Procedure Call
LAMP	Linux Apache MySQL PHP
MES	Manufactoring-Excecution-System
OLE	Object Linking and Embedding
OPC	OLE for Process Control
PHP	Hypertext Preprocessor
SCADA	Supervisory Control and Data Acquistion
SPS	Speicherprogrammierbare Steuerung
SQL	Structured Query Language
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
USB	Universal Serial Bus
URL	Uniform Resource Locator
VPN	Virtual private network
WAMP	Windows Apache MySQL PHP

1.4 Abgrenzung des Umfeldes

1.4.1.1 Feldebene

Einer bestehenden funktionsfähigen SPS-Heizungssteuerung soll eine HTTP-Client Kommunikationsschnittstelle implementiert werden. Die Anwendungsschnittstelle stellt dabei die bestehende Applikation dar. Um eine Kommunikation mit dem Webserver zu ermöglichen, wird eine Submenge des HTTP–Protokolls implementiert. Das Saia-Betriebssystem stellt das dafür nötige TCP/IP-API zur Verfügung. Das Kommunikationsmodul wird in die bestehende Prozesssteuerung integriert.

Die Software der Prozessablaufsteuerung ist bestehend, ebenso die Hardwareinstallationen. Neu hinzu kommt das Kommunikations Modul.

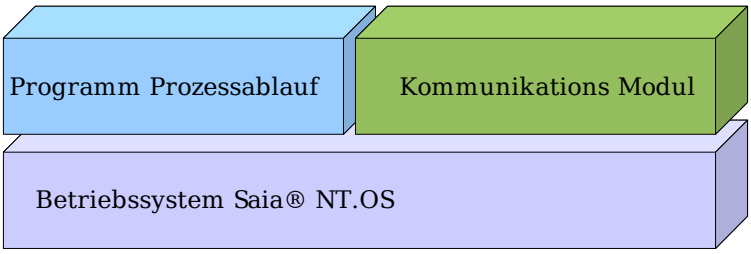


Abbildung 6: Abgrenzung Feldebene

1.4.1.2 Datenverarbeitungsebene

Auf die Apache Webserver-Installation,-Konfiguration und-Funktionen wird nicht näher eingegangen. Werden spezielle Einstellungen benötigt, werden diese erwähnt . Es ist jedoch eine Standardkonfiguration anzustreben. Dasselbe gilt für die MySQL-Datenbank. Ein installiertes PHP- und MySQL- Modul gilt als Voraussetzung. Auf Funktionen und Mechanismen des Webservers wird nur eingegangen, wenn es dem Verständnis von Abläufen oder Problemen dient.

1.4.1.3 Sicherheit

Ein wichtiger Punkt bei solchen Kommunikationen ist die Datensicherheit und Integrität, speziell wenn die Kommunikation übers Internet abgewickelt wird. Die Sicherheit ist jedoch nicht Inhalt dieser Arbeit. Das verwendete Gerät unterstützt HTTPS leider nicht, und zusätzlich noch ein Verschlüsselungsverfahren zu entwerfen, würde den Rahmen dieser Arbeit sprengen.

1.5 Feldebene

1.5.1 Installation

Um die Kommunikationssoftware auf der SAIA-SPS zu installieren, muss der Quellcode sowie die Programmierumgebung der Firma SAIA vorhanden sein.

Um die Datenverarbeitungsebene zu installieren, wird ein WAMP/LAMP-System vorausgesetzt. Eine automatische Installationsroutine ist nicht vorgesehen, die Installation ist jedoch nicht sonderlich kompliziert. Der PHP-Quellcode wird ins HTTP-Verzeichnis des Webservers kopiert. Das Datenbankschema wir bei einem Request des Feldgerätes automatisch erzeugt, wenn im Konfigurationsfile $globCfg['createTables'] = true gesetzt ist. Im File db_connection.php müssen die Datenbankverbindungs-Parameter eingetragen werden

1.5.2 Selbsttest

Der SPS-Befehlssatz stellt verschiedene Prüfroutinen zur Verfügung. Diese sind jedoch nicht Teil der Aufgabe und werden hier nicht näher erläutert.

1.5.3 System Setup

Zum System-Setup gehört die URL sowie die IP-Adresse des Webservers. Letztere kann über die Serviceschnittstelle (Abbildung Service Panel Kommunikation S.49)definiert werden. Die URL konnte leider nicht auf die Serviceschnittstelle geführt werden, weil das Verwendete Service-Tool die Kommunikation von Texten nicht unterstützt. Hier müsste sicher noch eine bessere Lösung gefunden werden. Die URL ist im Code einfach anzupassen. Bei einer IP-Adresse Änderung des Webservers sind also keine Codeanpassungen nötig. Vorteilhafter wäre es, wenn der DNS-Dienst zur Auflösung der IP-Adresse verwendet werden könnte. Das Feldgerät unterstützt DNS leider nicht.

1.5.4 Systemstart und System-Shutdown

Das System auf der Feldebene startet selbstständig, sobald der Strom eingeschaltet wird. Abgeschaltet wird es mit dem Ausschalten des Stroms. Der Webserver soll hier nicht behandelt werden, dies ist Sache des Hosters.

1.5.5 Verhalten bei Power Fail

Da der Power Fail das normale Shut Down-Prozedere der SPS ist, gehen dabei auch keine Daten verloren. Der letzte Datenstand wird vom Betriebssystem ins Flash der SPS geschrieben.

1.5.6 Sprachen

Die Sprache ist deutsch in den Kommentaren im Code jeweils Englisch.

1.5.7 Hardware

Die Hardware der Feldebene besteht aus einer SAIA SPS PCD2.M480 mit einem Freescale (vormals Motorola) Coldfire 5407 32-Bit CISC Prozessor. Als Erweiterung ist ein Ethernetmodul gesteckt.

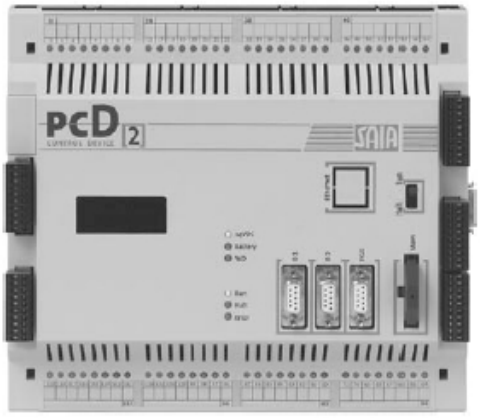


Abbildung 7: PCD2.M480

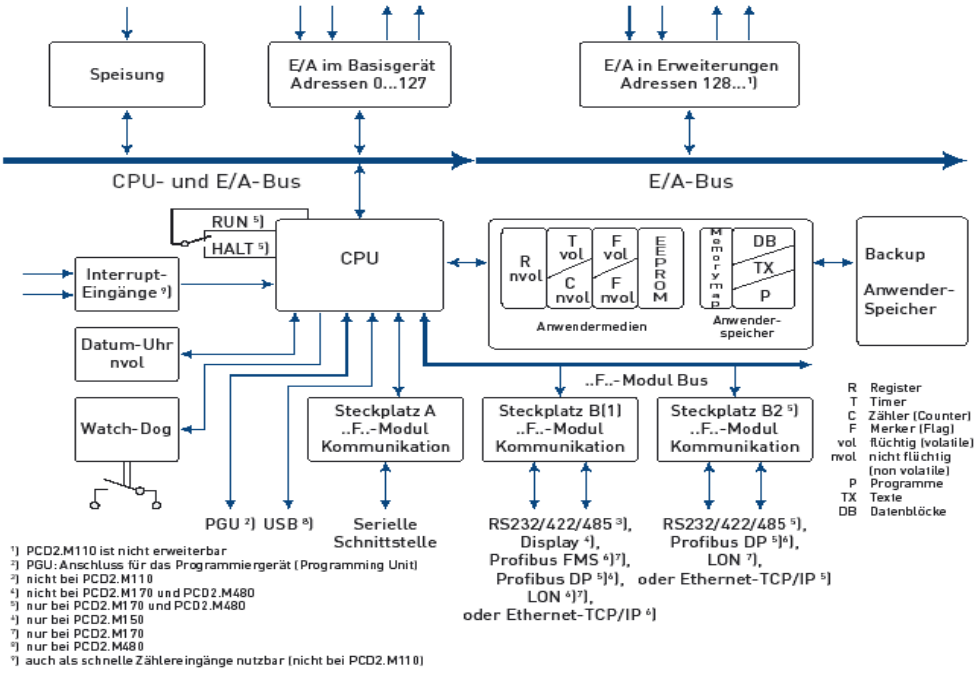


Abbildung 8: Blockschema PCD2.M480 (Benutzerhandbuch Saia)

Auf die PCD2.M480 kann seriell über RS232 Schnittstelle, über USB oder via Ethernet zugegriffen oder diese programmiert werden.

1.5.8 Programmierung

Die PCD2.M480 wird mit Anweisungsliste programmiert. Die Anweisungsliste, kurz AWL genannt, ist eine in IEC 61131-3 standardisierte Methode zur Programmierung von speicherprogrammierbaren Steuerungen (SPS).

Als Entwicklungsumgebung wird das von Saia mitgelieferte PG5, Version 1.3.122, verwendet. Abbildung unten: ein Screenshot des Editors mit einem Stück AWL -Code. Zu sehen ist die Definition eines Funktionsblocks. Unter einem Funktionsblock ist eine Softwarefunktion im gängigen Sinn zu verstehen. Die Parameter können nur „by reference“ übergeben werden.

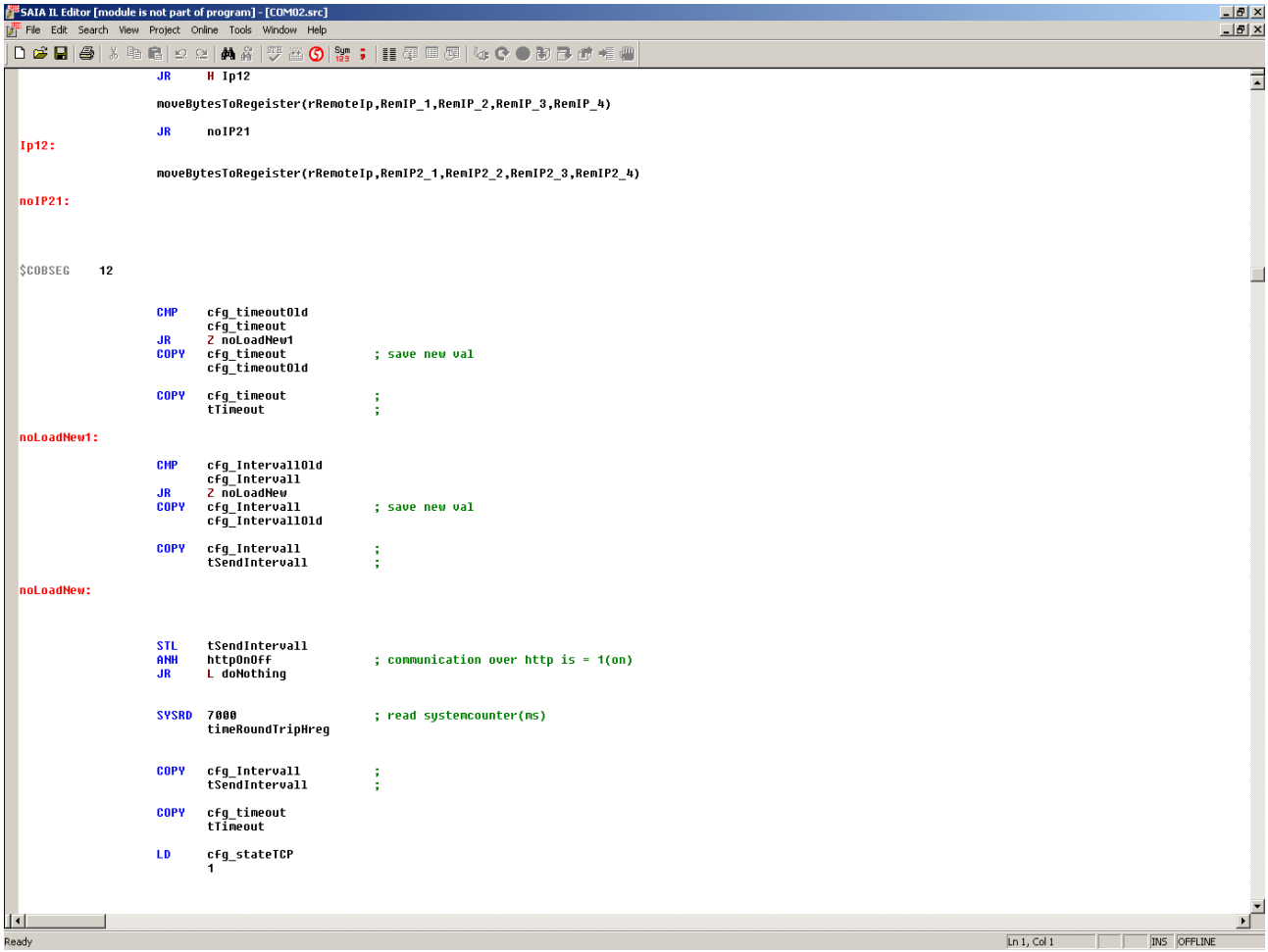


Abbildung 9: Programmierumgebung Saia

Die Sprache ist an Assembler angelehnt. Der Code wird in so genannte Ausführungsblöcke strukturiert. Saia nennt dies BLOCTEC. BLOCTEC ist eine Methode zur Programmstrukturierung, d.h. zum Unterteilen von Anwenderprogrammen in einzelne, eigenständige Programmteile, sog. Blocks.

Der **zyklische Organisations-Block (COB)** ist das Grundstrukturelement eines Programms. Vergleichbar mit der Main Funktion in C. Mit dem Unterschied, dass bis zu

16 COB's definiert werden können. Ein COB ist im eigentlichen Sinn ein Prozess. Dies ermöglicht den Aufbau eines kooperativen Multitaskings. Auch wenn ein COB noch nicht vollständig abgearbeitet ist, kann er seine Rechenzeit jederzeit an den nächsten COB abgeben. Mit dem Befehl NCOB wird dies bewerkstelligt. Weiter ist es möglich, ein COB in den idle Zustand zu setzen. Mindestens ein COB muss vorhanden sein, damit ein Anwenderprogramm lauffähig ist. Aus einem COB werden im Normalfall kleinere Programmeinheiten wie **Programm-Blocks (PB)** und **Funktions-Blocks (FB)** aufgerufen. PBs und FBs können wiederum andere PBs und/oder FBs aufrufen, bis zu einer Verschachtelungstiefe von sieben Ebenen (acht mit dem COB).

Die **Programmvariablen** werden in so genannten Registern oder Flags gespeichert. Register sind 32-Bit Ganzzahl oder Float-Werte, je nachdem welche Befehle darauf angewendet werden. Flags sind 1-Bit-Speicherstellen. Desweiteren stehen dem Entwickler sogenannte Datenblöcke zur Verfügung. Sie können als eindimensionales Array von 32-Bit-Werten definiert werden. Ein Datenblock kann maximal 16384 Elemente aufnehmen. Während auf Register Rechenoperationen direkt ausgeführt werden können, muss das Element eines Datenblockes zuerst mit einem zusätzlichen Befehl auf ein Register kopiert werden, will man damit rechenoperationen ausführen. Das Arbeiten mit Datenblöcken ist also etwas umständlicher als Registermanipulationen. Datenblöcke sind die Massenspeicher-Elemente der Saiasteuerungen. Man kommt daher nicht umhin, mit denselben zu arbeiten. Die Saiasteuerungen kennen folgende Ressourcetypen:

Resource	Beschreibung	Adress-Bereich
I	Input (Eingang)	0 – 8191
O	Output (Ausgang)	0 – 8191
F	Flag (Merker)	0 – 8191
T	Timer (Zeitglied)	0 – 450
X	Text	0 – 3999
DB	Data Block	0 – 3999

Der Saia-Präprozessor lässt auch **Makrokonstrukte** zu, wie dies herkömmliche C-Präprozessoren auch tun. Makros haben eine **Reihe von Vorteilen** gegenüber den Funktionsaufrufen (FB) bei Saia Steuerungen. Dasselbe Makro kann mit unterschiedlicher Anzahl Parameter aufgerufen werden, ein entscheidender Vorteil für viele Arten von Anwendungen. Variablen können bei Saia Funktionen(FBs) nur „by reference“ übergeben werden. Daher ist das Programmverhalten bei einem Makro -Aufruf identisch mit einem Funtionsblock-Aufruf, da Makroparameter immer eine syntaktische Referenz von einem Variablennamen sind. Weiter können in Saia- Funktionen die Adressen der mitgegebenen Variablen nicht mehr eruiert werden, gerade diese werden aber öfters benötigt. Desweiteren können mit Saia FBs keine lokalen Variablen deklariert werden, in Makros hingegen schon. Auch lassen sich Namensräume mithilfe der Saia-Makros konstruieren, wie man das in späteren Kapiteln sehen kann. In diesem Projekt wurden daher vielfach Makros anstelle von Funktionsblöcken verwendet. Makros laufen im übrigen auch etwas schneller ab. Das Kompilat des Programmcodes wird natürlich mit häufiger Verwendung von Makros etwas grösser.

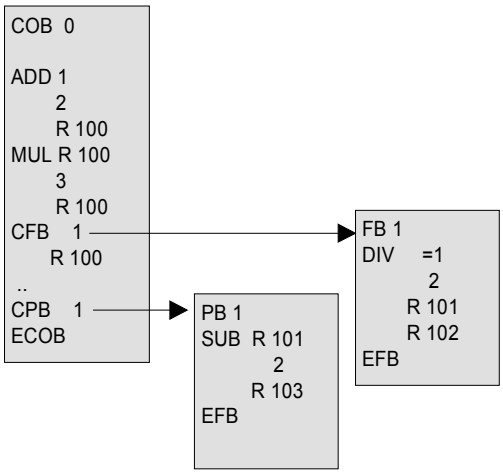


Abbildung 10: Prgrammbeispiel in Bloctec

In Abbildung 10 ein Beispiel eines AWL-Programmes. COB 0 wird gestartet. 1 mit 2 addiert und im Register 100 gespeichert. Register 100 wird mit 3 multipliziert, im Register 100 ist jetzt 9, danach wird der Funktionsblock 1 mit Register 100 als Parameter aufgerufen. Im FB 1 wird der Parameter 1 (R 100) mit 2 dividiert, das Resultat in Register 101 abgespeichert (Ganzzahldivision im Register 101 steht jetzt 4), der Divisionsrest wird im Register 102 gespeichert. Danach wird im COB 0 der Programmblock 1 aufgerufen. Dort wird 2 vom Register 101 subtrahiert, und in Register 103 abgespeichert. Danach werden, falls vorhanden andere COBs aufgerufen ansonsten startet das Programm von vorne.

Hier wird ersichtlich dass sich etwas komplexere Programmgebilde sehr schnell über sehr viele Zeilen erstrecken. Es ist daher eine besondere Herausforderung, übersichtliche und lesbare Programme zu schreiben. Programmverzweigungen und Schleifen, die mit Jumpes realisiert werden müssen, vereinfachen die Lesbarkeit der Programme nicht gerade. Es empfiehlt sich daher die Programme überlegt zu strukturieren.

1.6 Datenverarbeitungsebene

1.6.1 Umgebung

Die auf der Feldebene anfallenden Daten werden auf einem Webserver gespeichert und verarbeitet. Als Webserver wird ein so genanntes LAMP-oder WAMP System verwendet. LAMP/WAMP Webserver-Systeme bestehen aus einem Server mit einem Linux- oder Windows-Betriebssystem und einem darauf aufsetzenden Apache-Webserver. Der Apache-Webserver beinhaltet unter anderem ein PHP-Modul, dieses wiederum verfügt über eine MySQL-Datenbank-Schnittstelle. Der Vorteil in der Verwendung eines solchen Systems liegt vor allem in deren Verbreitung bei kommerziellen Hostern und in dessen kostenlosen Verfügbarkeit. Damit die Datenbankschemata und PHP Programme dieser Arbeit funktionieren, wird MySQL und PHP ab der Version 5.0 vorausgesetzt.

1.6.2 Programmierung

PHP ist eine objektorientierte Skriptsprache mit deren Hilfe sich dynamische Webseiten erstellen lassen. PHP ist syntaktisch stark an C angelehnt. So existiert zum Beispiel auch der allseits bekannte printf-Befehl. PHP zeichnet sich aus, durch eine grosse Funktionsbibliothek. Alleine zum Bearbeiten von Arrays existieren über 40 Funktionen. PHP ist zur Zeit in Version 5.3 als Alpha-Version verfügbar. Auf www.php.net findet sich mehr Informationen zum Thema.

Als Entwicklungsumgebung kann ein beliebiger Editor mit oder ohne Syntax- Highlighting verwendet werden. Um umfangreichere Projekte zu verwalten, würde sich ein Plugin für Eclipse anbieten. Zumal für Eclipse Versions-Verwaltung-Tools wie CVS zur Verfügung stehen. In vorliegender Arbeit wird das Zendstudio verwendet.

1.6.3 Datenbank

Als Datenbank bietet sich wie bereits erwähnt der MySQL Server in der Version > 5.0 an. Dieser zeichnet sich gegenüber seinen Vorgänger-Versionen mit erhöhter Funktionalität aus. So lassen sich sich Stored Procedures, Views und Triggers definieren. Auch Transaktionen können verwendet werden.

1.6.4 Benutzereigenschaften

Um Zugriff auf die Benutzeroberfläche der Datenaverarbeitungseben zu erhalten muss Benutzername und Passwort eingegeben werden.

Damit wird der Benutzer identifiziert, und es werden ihm die entsprechenden Rechte zugewiesen.

1.7 Systementscheide

Die auf dem Feldgerät entwickelte HTTP Schnittstelle kann Verbindung zu beliebigen Webservern herstellen, wenn auf dem Webserver die Daten dem definierten Protokoll entsprechend ausgewertet werden. Als Muster Implementation wurde ein Webserver-Datenbanksystem gewählt, das die grösste Verbreitung hat, PHP und MySQL. Dieses Konfiguration ist beinahe bei jedem Webhoster zu günstigen Konditionen verfügbar.

1.8 Verwendete Software und Tools

Apache/2.2.0
PHP Version 5.2
MySQL 5.0.18
MPHMyAdmin Version 2.7.0
Firefox 2.0.0.16 mit Webdevelopper Plugin 1.1.6
Microsoft Internet Explorer 6.0.29
Wireshark Version 0.99.3
Zend Studio Version 3.5.2
Saia PG 5 Version 1.3
Openoffice Version 2.4
Microsoft Excel
ProMoSNT Version 1.3
Jude Community Version
3.1 Micrografx 6.0.b
UltraeEdit-32 Version 9.2
Textpad 4.7.2
Total-Commander 5.5

2 Wahl der Übertragungsart

2.1 Wahl des Übertragungsprotokolls

Eine der Schwierigkeiten beim Programmieren von Mikrocontrollern ist in vielen Fällen die Knappheit der Ressourcen. So auch in unserem Fall. Man möchte möglichst viel Rechenzeit für die Anwendung zu Verfügung haben und möglichst wenig für die Kommunikation verbrauchen müssen. Daher ist eine Übertragungsart gefragt, die möglichst wenig Ressourcen und Prozessorzeit beansprucht.

Will man über das Internet kommunizieren können ist das IP-Protokoll gegeben. Möchte man ohne spezielle Konfiguration der Firewall über diese hinweg kommunizieren können, so wird am besten das meist verbreitetste Protokoll verwendet, welches Firewalls meist unbehelligt passieren kann, nämlich dass HTTP Protokoll. Möchte man zusätzlich die eingangs erwähnte Effizienz berücksichtigen, empfiehlt sich, ein HTTP-File-Upload zu verwenden. Mit dieser Methode können die Feldgeräte Binärdaten direkt ohne aufwändige Konvertierung in einen Buffer schreiben, welcher dann zyklisch gesendet wird. Dies wäre eine asynchrone Kommunikation. Die Konvertierung würde der vergleichsweise rechenstarke Webserver übernehmen. Deterministisch ist diese Art der Kommunikation natürlich nicht. Für Logging, Trenddatenerfassung und Überwachung von Anwendungen ist dies in den meisten Fällen genügend.

2.2 Beteiligte Kommunikationspartner

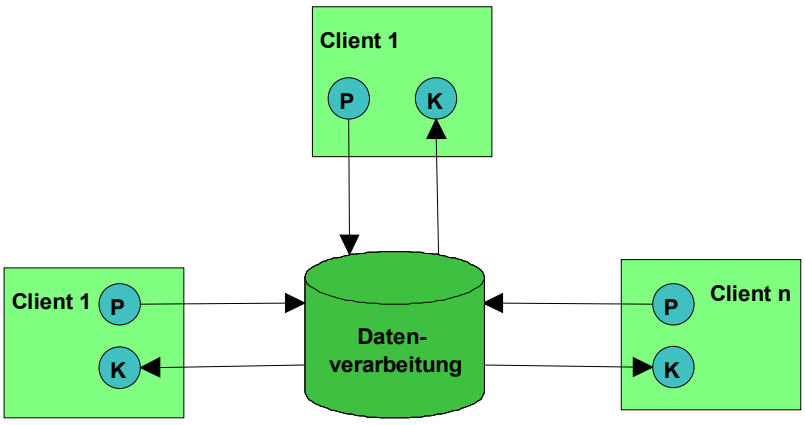


Abbildung 11: Kommunikationsstruktur mit Zentraler Datenverarbeitung

Ist die Technologiefrage geklärt, stellt sich als nächstes die Frage wie die Technologie einzusetzen ist. Es gibt verschiedene Möglichkeiten, die Daten zu senden. Die Kommunikationsstruktur besteht aus einer zentralen Datensenke und den darum herum angeordneten Clients. Jeder Client besteht aus einem Datenkonsumenten (K) und einem Datenproduzenten (P). Gelangen Daten von einem Produzent zur Datenverarbeitungsebene, so muss diese die Daten speichern und an alle anderen Clients bzw. an deren Konsumenten verteilen.

2.3 Datenclients

In unseren Anwendung gibt es zwei Arten von Clients: Feldebene-Clients und Bedienebene-Clients. Beiden sind ihre spezifischen Eigenheiten gegeben durch Rechenstärke, verfügbarem Arbeitspeicher und dessen Verbindung zur Datenverarbeitungsebene. Gemeinsam haben die beiden Clients das HTTP–Protokoll welches sie mit der Datenverabeitungsebene verbindet.

Alle Daten, die gesendet werden, müssen vom Empfänger auch wieder identifiziert werden können. Dazu werden ids mitgesendet. Diese ids oder Kennungen sind die Overhead Daten. Die Grösse der Kennung, die verwendet werden muss, ist abhängig davon wie viele unterschiedliche Kategorien bzw. Pakete man bilden möchte. Hat man 32-Bit Werte und ein Adressraum von 32 Bit, den man übermitteln möchte, so ergibt sich eine 32 bit lange id.

Wählt man eine kleine Kennung und eine im Verhältnis grosse Datenmenge so entsteht ein geringer Overhead. Im Prinzip ist es eine Frage nach der Grösse der Kennung und der ihr zugeordneten Datenmenge.

Die am Bandbreitensparendste Art zu kommunizieren ist je nach Datenaufkommen die Push Variante. D.h. alle Produzenten schreiben bei Bedarf die Daten in die Datenverarbeitungsebene, diese leitet die Daten an die anderen Clients weiter. Wir sprechen hier also von einer Ereignisorientierten und synchronen Kommunikation. Diese wäre mit den heutigen Browsern und den gängigen Firewall aufbauten nur umständlich zu lösen. Firewalls müssten speziell konfiguriert werden, bzw Ports geöffnet werden, sowie Plugins oder Applets für Browser müssten entwickelt werden. Meist ist die Einflussnahme auf die IT-Infrastruktur bei Firmen beschränkt, will man Geräte mit Aussenweltverbindung ohne Zusatzaufwand in bestehende Netzwerke installieren, so ist es problemloser auf die verbreitete Pull Variante (wie ein Web-Browser dies tut) zurückzugreifen. Ein Browser sendet bei Bedarf eine Anfrage (Request) für Daten zum Server. Dieser sendet die nachgefragten Daten mit der Response umgehend an den Client zurück.

2.4 Varianten

Wie oben begründet ist eine ereignisorientierte Kommunikation zu umständlich. Aber auch bei der Client-Pull (oder Polling) Variante gibt es verschiedene Realisierungsmöglichkeiten. Aus Sicht des Clients lassen sich verschiedene Kommunikationsvarianten unterscheiden:

Write Single - Read Single

Nur geänderte Werte werden zur Datenverarbeitungsebene gesendet, nur geänderte Werte werden von der Datenverarbeitungsebene gelesen.

Write All - Read Single

Es werden alle Werte an die Datenverabeitungsebene gesendet aber nur die geänderten gelesen.

Write Single - Read All

Es werden nur die geänderten Werte zur Datenverabeitungsebene gesendet, aber alle gelesen.

Write All - Read All

Es werden alle Werte zur Datenverarbeitungsebene gesendet und alle Werte von der Datenverarbeitungsebene gelesen.

Um geänderte Werte festzustellen, müssen die zu kommunizierenden Werte mit den Werten aus dem letzten Kommunikationszyklus verglichen werden. Unterscheiden sie sich, muss der Wert gesendet werden. Dies benötigt Speicherplatz und Rechenzeit. Für die zu sendenden Daten muss eine Id vergeben werden, was weiteren Speicherplatz belegt und zusätzlich den Daten-Overhead vergrössern (je nachdem sogar verdoppeln) würde. Diesen Nachteil haben alle „Write-Single“-Varianten gemeinsam.

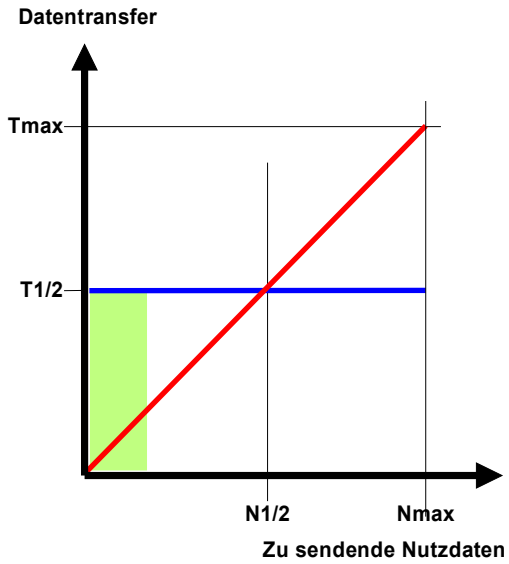


Abbildung 12: Datentransfer-Nutzdaten

Zeichnet man qualitativ den Datentransfer zu den Nutzdaten, die gesendet werden

müssen, in einem xy-Diagramm auf, so ist ersichtlich, dass (solange weniger als die Hälfte der Gesamtdaten gesendet werden) es günstiger ist mit einem Id-Value System zu fahren. Vorausgesetzt Id und Nutzdateneinheit haben die gleiche Grösse. Die blaue Linie zeigt die Write-All-Variante, die rote Linie die Write-Single-Variante. Die Write-All-Variante wird vom Datentransfervolumen her erst dann interessant, wenn die Werte häufig ändern und somit in jedem Kommunikationszyklus gesendet werden müssen. Ändern sich die Daten relativ selten, benutzt man besser die Write-Single-(Id-Value)-Variante. In dieser Anlage schätze ich, dass ein grosser Teil der Daten über längere Zeiträume konstant bleiben und daher selten gesendet werden müssen (wir bewegen uns daher im Diagramm im grün eingefärbten Bereich). Auf dieser Annahme basiert die folgende Vergleichstabelle.

Kommunikationsarten vom Client aus gesehen.

	Bandbreite	Rechenzeit Feld-Client	Rechenzeit Browser-Client	Rechenzeit Datenverabeitung
Write Single Read Single	klein	gross	klein	klein
Write All Read Single	mittel	**klein**	normal	mittel
Write Single Read All	mittel	mittel	**klein**	mittel
Write All Read All	gross	klein	klein	mittel

Grundsätzlich sollen nur geänderte Werte in die Datenbank geschrieben werden. Sonst werden unter Umständen von anderen Clients, inzwischen geänderte Werte überschrieben. Daher wird je nach Kommunikationsvariante die Rechenzeit und Speicherbedarf, um die geänderten Werte zu eruieren, von einem Kommunikationspartner zum anderen verschoben. Der Rechenaufwand beim Client ist für die Write-All Variante daher gering weil die Vergleichsarbeit von der Datenverarbeitungsebene übernommen wird.

Aus der obigen Gegenüberstellung geht hervor, dass die Write-All Read-Single Variante auf dem Client am wenigsten Rechen-und Speicherbedarf einnimmt und somit als die beste Variante angesehen werden kann, vorausgesetzt es steht genügend Bandbreite zur Verfügung. Wie gross ist denn überhaupt dass zu erwartende maximale Kommunikationsvolumen? Die Frage wird im nächsten Kapitel beantwortet.

2.4.1 Datenaufkommen

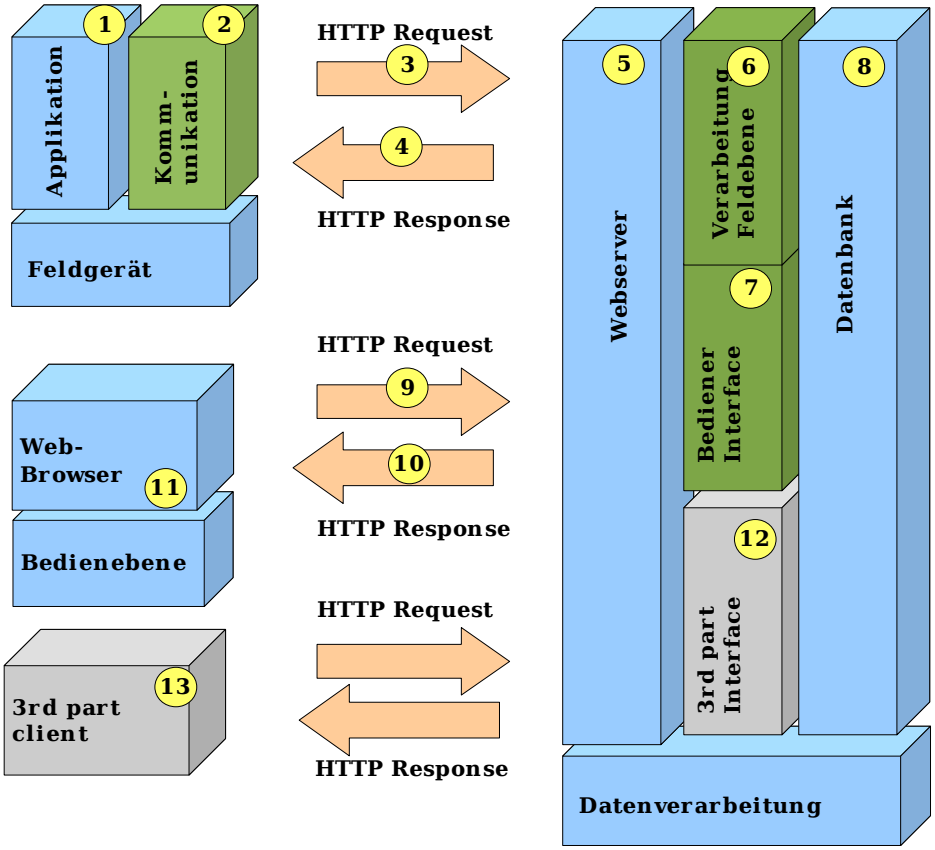
Auf dem jetzigen Client sind rund 1100 32 Bit Integer-Werte zu kommunizieren. Dazu gesellen sich rund 1200 Flags (Booleans).

Total in Bytes: 1100*4 + 1200/8 = 4550 Bytes . Rechnen wir noch den Overhead dazu , ergeben sich ca. 7-10 Kbyte pro Request. Die TCP/IP MTU, welche die Saia Ethernetschnittstelle verarbeiten kann ist 744 Bytes.

Dadurch wird klar, dass sich eine ereignisgesteuerte Kommunikation bei einer Breitbandverbindung nicht lohnt. Im Gegenteil: Die Durchlaufzeit auf dem Feldgerät würde sich erhöhen und damit die Kommunikationsgeschwindigkeit sinken. Zudem würden sich kleinere Transporteinheiten kaum lohnen, da das Verhältnis Nutzlast Overhead sich (viel) ungünstiger entwickeln würde, je kleiner die Nutzdatenpakete werden. Die Kommunikationsgeschwindigkeit ist bei so geringem Datenvolumen in erster Linie von der Antwortzeiten des Servers abhängig.

3 Systemübersicht

Die aus dem Kapitel 2.1, Wahl des Übertragungsprotokolls, hervorgegangene Kommunikationsvariante soll nun beschrieben werden. Unten eine Übersicht des Systems. Die einzelnen Positionen werden in den folgenden Kapitel genauer erläutert.



1. Applikationssoftware Feldgerät

Die Applikationssoftware steuert die Anwendung des Gerätes. Sie tauscht mittels Methoden der Kommunikationssoftware applikationsspezifische Daten aus.

2. Kommunikationssoftware Feldgerät

Die Kommunikationssoftware sendet in regelmässigen Abständen Daten zum Server (Push Betrieb) und erhält Daten mit der Response vom Server.

3. HTTP-Request zum Server Kap.4, S.26

Der Request beinhaltet das Prozessabbild des Feldgerätes in Form eines HTTP–File uploads nach RFC 1867.

4. HTTP-Response vom Server

Der Response enthält die Daten für den Client in Form eines binären HTTP File-Downloads.

5. Webserver

Der Webserver nimmt die HTTP Requests der Clients entgegen.

6. Verarbeitung Feldebene Kap. 5, S.29

Verarbeitet die vom Webserver empfangenen Daten und schreibt diese bei Bedarf in die Datenbank.

7. Bediener Interface Kap. 8, S.53

Liest und schreibt Daten aus der Datenbank und stellt sie dem Benutzer tabellarisch oder Graphisch dar.

8. Datenbank Kap7, S.52

Speichert die Prozess-und Historydaten.

9. Request Browser

Beinhaltet die Sessiondaten bzw die geforderte Ansicht der Daten.

10. Response Browser

Enthält die darzustellenden Daten, bestehend aus HTML, CSS und Javascript.

11. Browser

Interpretiert den vom Server im Response gesendeten HTML Code und stellt ihn dar.

12. 3rd partie Interface

Auch einem Fremdsystems könnten die Daten verfügbar gemacht werden Die Schnittstelle müsste an die Bedürfnisse des Fremdsystems angepasst werden. Eine Möglichkeit dies zu tun wäre zum Beispiel mit Webservices, also mit SOAP oder RPC.

13. 3rd partie Applikation

Datenkonsument oder Lieferant eines Fremdsystems. Third-partie Clients sind im Moment kein angebunden.

3.1 Datenfluss Feldgeräte – Datenverarbeitung - Browser

Die Regel-und Steueraufgaben der Applikation sind jeweils einzelnen Funktion zugeordnet. So sind als Beispiel die Messwerterfassung und die Regelung jeweils einer Funktion zugeordnet. Der Eingangsvektor (Parameter) der jeweiligen Funktion beinhaltet alle wichtigen Informationen über den Zustand des Objektes(Funktion).

Somit kann auf der Datenverarbeitungsebene einer Funktion jeweils eine Tabelle zugeordnet werden. Der Eingangsvektor einer Funktion wird in der Tabelle als Datensatz abgebildet.

Im Browser wiederum lässt sich der Inhalt der Tabellen auf übersichtliche Weise darstellen.

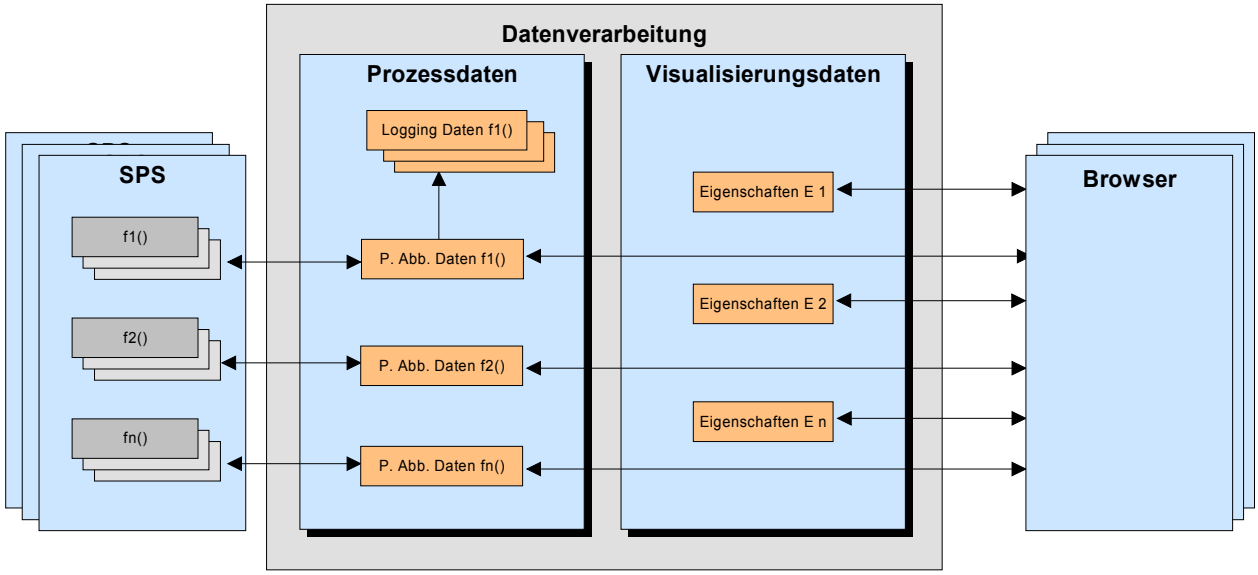


Abbildung 13: Datenfluss Feldebene-Datenverarbeitung

4 HTTP-Request zum Server

4.1 Design

Um die Daten vom Feldgerät der richtigen Datenbank, der richtigen Tabelle, dem Richtigen Datensatz und dem richtigen Datenfeld zuordnen zu können, müssen die Daten entsprechend organisiert gesendet werden. Jedem Feldgerät, jeder Funktion, jedem Funktionsaufruf und jedem Parameter in der Feldebene wird deshalb eine eindeutige ID zugeordnet.

Die ID vom Feldgerät entspricht einer Datenbank, die ID einer Funktion entspricht einer Tabelle, die ID eines Funktionsaufrufs entspricht der Datensatz ID und die Parameter ID entspricht einem Datenfeld in einer Tabelle. Zusammengefasst:

Feldgerät		Datenverarbeitung
Gerät	=	Datenbank
Funktion	=	Tabelle
Funktionsaufruf	=	Datensatz
Parameter	=	Datenfeld

Da nun die Umsetzung der Daten von der Feldebene zur Datenverarbeitungsebene bekannt ist, muss als nächstes festgelegt werden, wie die Daten von der Quelle zum Ziel gelangen. Es stellt sich die Frage wie das Modell effizient auf das HTTP–Protokoll umgesetzt werden kann.

Die ID des Feldgerätes kann in ein HTTP-Header-Feld geschrieben werden. Da ein File–Upload -Request mehrere Files enthalten darf, kann die ID einer Funktion dem Filnamen bzw. dem Formularnamen zugeordnet werden. Um nicht unnötigen Overhead zu produzieren werden die Daten der Parameter zusammen mit der Funktionsaufruf-ID serialisiert und binär in das hochzuladene File geschrieben:

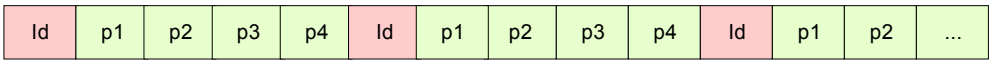


Abbildung 14: Inhalt eines hochgeladenen Files

Dem Inhalt der Datenfelder (od. Parameter) wird immer die Datensatz ID (bzw. Funtionsaufruf ID) vorangestellt. Die ID der Parameter ergibt sich aus der Reihenfolge der Parameter. Auf dem Server muss definiert werden, welcher Parameter in welches Feld der Tabelle geschrieben werden muss. Diese Zuordnung kann in einem Config–File definiert werden.

Wenn man nun das Ergebnis der Überlegungen anschaut, erhält man das folgenden Konstrukt für den Http-Request:

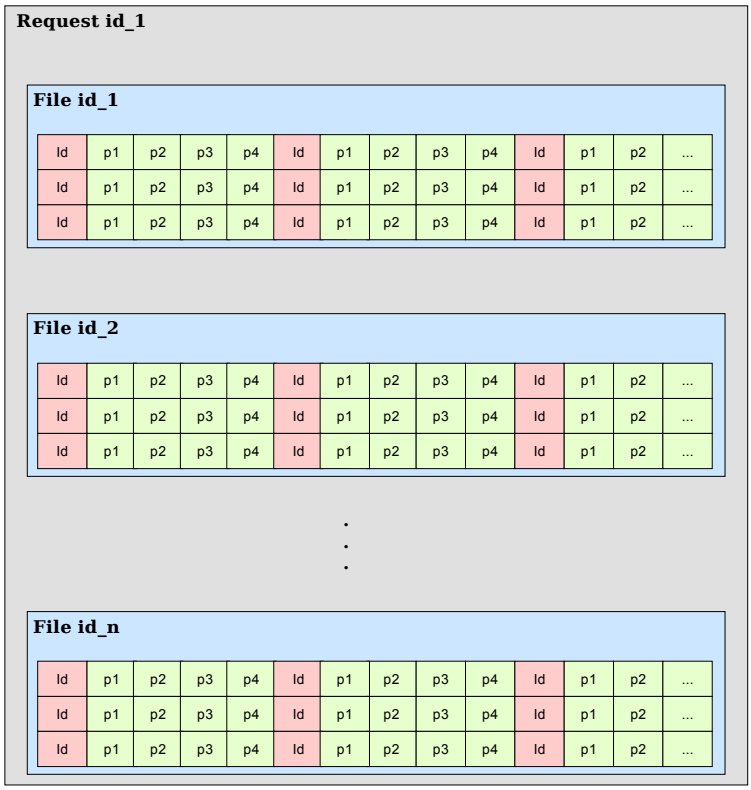


Abbildung 15: Organisation der Daten pro Request

Gut ersichtlich ist der Eingangsvektor einer Funktion jeweils beginnend mit der ID und den darauf folgenden Daten.

4.2 Umsetzung

Eine HTTP-Nachricht ist zweiteilig gegliedert und besteht aus Header und Body. Genauere Defintionen und Beschreibungen finden sich in RFC 2616. Für das geplante Vorhaben kommt die POST Methode zur Anwendung. Der Header muss dann wie folgt ausehen:

POST /pv/rec.php
HTTP/1.1 Content-Type: multipart/form-data; boundary=8a14204d2c598
Host: www.anderegg.org
Content-Length: 4902

In der ersten Zeile wird die Methode, wie mit dem Server kommuniziert wird, spezifiziert. Des weiteren muss das Zieldokument, welches den Request verarbeiten soll, angegeben werden. Im Feld Content-Type muss der Inhalt des Requests spezifiziert werden (nach RFC 2388). Das Feld Content-Length beinhaltet die Länge des Bodys in Bytes. Im Feld Host muss die gewünschte URL definiert werden, damit der Server, falls dieser mehr als eine Domain hostet, die Daten richtig zuordnen kann.

Im Body werden die einzelnen Files mit der im Header angegebenen boundary unterteilt. Die boundary muss so gewählt werden, dass ein Teil des binären Inhalts des Files nicht zufällig die gleiche Form wie die boundary selbst annehmen kann. Den boundarys werden zwei „--“ vorangestellt. Zwischen den File-boundarsy's sind die

beiden Felder Content-Disposition und Content-Type aufzuführen. Content-Disposition enthält die Attribute form-data und filename. Form-data enthält den namen des Formulares und filename den Dateinamen. Darauf folgt der binäre Inhalt.

Abschnitt eines Files:

--8a14204d2c598
Content-Disposition: form-data; name="Da0001"; filename="id0001i.bin"
Content-Type: application/octet-stream .....
--8a14204d2c598

Der Filename ist mit idxxxxr.bin gewählt, für xxxx wird eine vier Zeichen lange Id verwendet, sie ist die Funktionsaufruf Id(bzw. Tabellen Id). Mit vier Zeichen kann eine genügend grosse Anzahl Id's generiert werden. Das „i“ am Schluss steht für eine 32 bit Integer Zahl. Damit weiss der Server wie lang und vor allem auf welche Weise der Binärstream zu verarbeiten ist. Zusätzlich ist noch der Typ „b“ für Bit implementiert. Weiter Datentypen, wie Floats, könnten noch implementiert werden.

Ist das letzte File angefügt muss der letzten boundary noch der Abschlusstring „--“ angefügt werden. Sie sieht dann wie folgt aus:

--8a14204d2c598--

5 Verarbeitung Feldebene

5.1 Design

Die Aufgabe der Verarbeitung-Feldebene ist es, die vom Webserver in den Http-Requests empfangenen Dateien vom Feldgerät (Client) auszulesen, zu verarbeiten und bei Bedarf in die Datenbank zu schreiben. Wie im Kapitel 2.4 Varianten S.20 Beschrieben, muss die Ereignisroutine auf dem Server implementiert werden. D.h. es muss eruiert werden können, ob sich ein Wert geändert hat. Hat er sich geändert, muss er in die Datenbank geschrieben werden. Würde das System als „only viewing“ ausgelegt, so könnten natürlich immer alle Werte in die Datenbank geschrieben werden. Da aber auch Daten an das Feldgerät gesendet werden, und somit noch andere Datenquellen Werte in die Datenbank schreiben, dürfen nur geänderte Werte in die Datenbank gelangen. Sonst würden die neu geschriebenen Daten von anderen Quellen, zum Beispiel Benutzereingaben, wieder überschrieben.

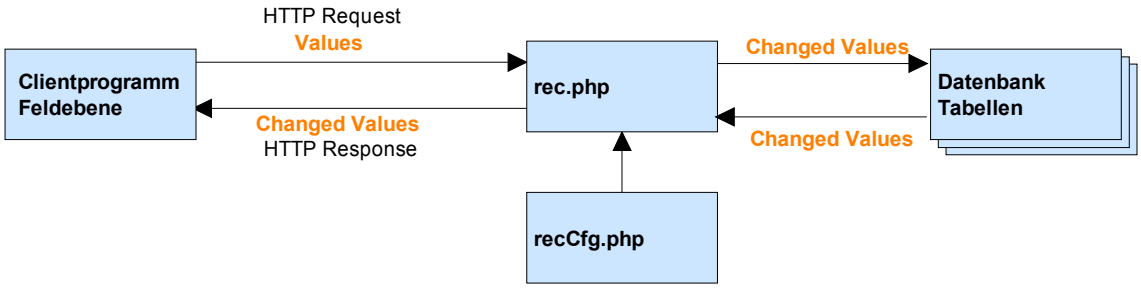


Abbildung 16: Datenfluss Feldgerät

Abbildung 16 verdeutlicht den Zusammenhang.

5.2 Umsetzung Schnittstelle zu Feldebene

Im File rec.php wird die Klasse fileToDB instanziert. Und die Methode WriteFileToDB der Klasse fileToDB in einer Schleife ausgeführt. Dabei wird jeder Eintrag, des assoziativen Arrays im Konfigurationsfile „recCfg.php“ einzeln ausgelesen. Ein Eintrag beinhaltet die Informationen wie ein vom Client hochgeladenes File zu behandeln ist:

```
$recCfg[] = array('table'=> 'hei01',
                  'fields'=> 'ID_HEI01,k_b,k_m,k_position,Ri,Rt,Temperatur,Ui,Uq',
                  'fieldsLog'=> 'Temperatur',
                  dataType'=> 'i',
                  'ModId'=> 9);
```

Zum Syntax: Die Variable $recCfg beschreibt ein numerisches Array das mit „[]“ automatisch mit jedem Eintrag erweitert wird. Jedem Eintrag wird ein Sub-Array mit einem Assoziativen Array zugewiesen. Der Inhalt des Assoziativen Arrays mit Schlüssel-Wertpaaren hat folgende Bedeutung:

table	In welcher Tabelle die Daten zu speichern sind.
Fields	Komma separiert die Feldnamen der Tabelle. Zuvorderst steht immer der Feldname des Primärschlüssels der Tabelle.
FieldsLog	Besagt welche Felder der in „fields“ angegebenen Felder kontinuierlich mit einem Zeitstempel versehen abgelegt werden sollen. Auch hier können die Felder Kommasepariert angegeben werden
DataType	Bestimmt wie die Daten in Länge und Format zu verarbeiten sind. „i“ steht für einen 32 bit integer im Format. Zur Zeit sind nebst dem „i“ Typ noch ein „b“ Typ für Booleans implmentiert. Die Typen konvertierung können nach belieben erweitert werden. Beispielsweise mit ein Float. Dazu müsste die Methode bitStreamToArray() der Klasse „FileToDB“ mit der gewünschten Transformationsfunktion ergänzt werden.
ModId	Hier wird die Modul Id angegebenen.

Die Verbindung, die durch das Konfigurationsfile „receive_conf.php“ zwischen der Datenbank-Tabelle und dem hochgeladenen File hergestellt wird, sieht wie folgt aus:

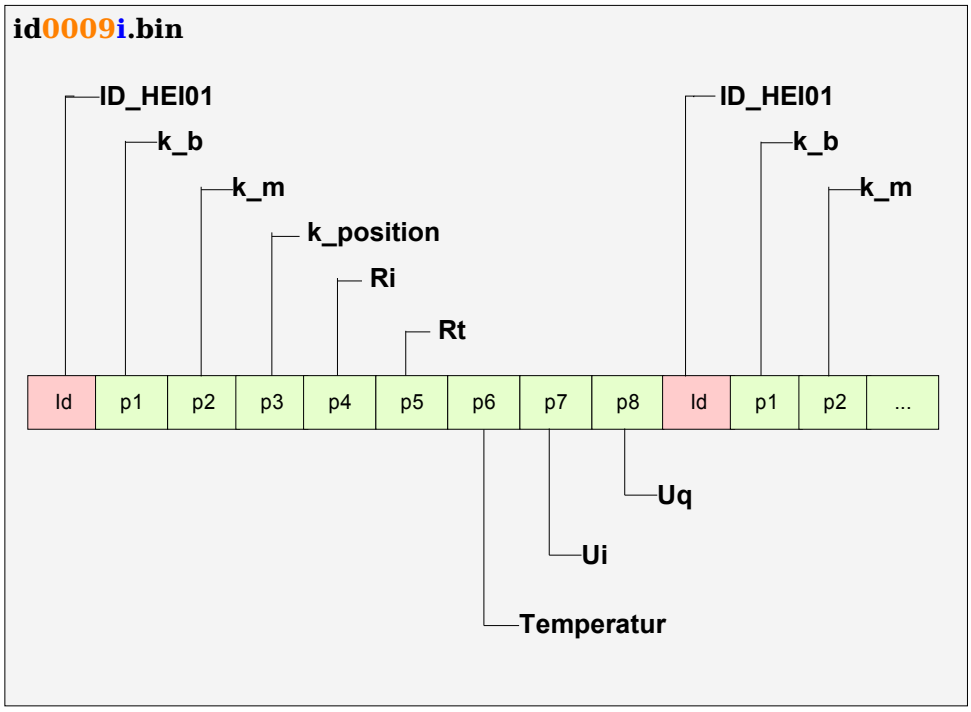


Abbildung 17: Interpretation eines hochgeladenen Files

Orange die „ModId“ integriert im Filenamen, blau der im File enthaltene Datentyp. Die Daten werden als kontinuierlicher Datenstrom eingelesen und Einheit für Einheit einem Datenbankfeld zugeordnet. Die Länge einer Einheit wird durch den Datentyp bestimmt. Die Art wie eine Einheit interpretiert wird, bestimmt ebenfalls der Daten-

typ. Das erste Element beinhaltet immer die Datensatz Id und wird immer als 32 Bit Integer im Big-Endian-Format interpretiert.

In Abbildung 18 der Ablauf des File-upload-Vorgang. Zuerst wird eine Verbindung zur Datenbank hergestellt. Schlägt der Verbindungsaufbau fehl, wird ein Eintrag ins Error.txt-File geschrieben und die Abarbeitung gestoppt. Steht die Verbindung zur Datenbank, wird ein Objekt der Klasse fileToDB instanziert. Die Methode writeFileToDB wird in einer Schleife aufgerufen. Pro Aufruf wertet Sie einen Eintrag im Array „$recCfg“ im Definitionsfile receive_conf.php und somit ein Hochgeladenes File aus.

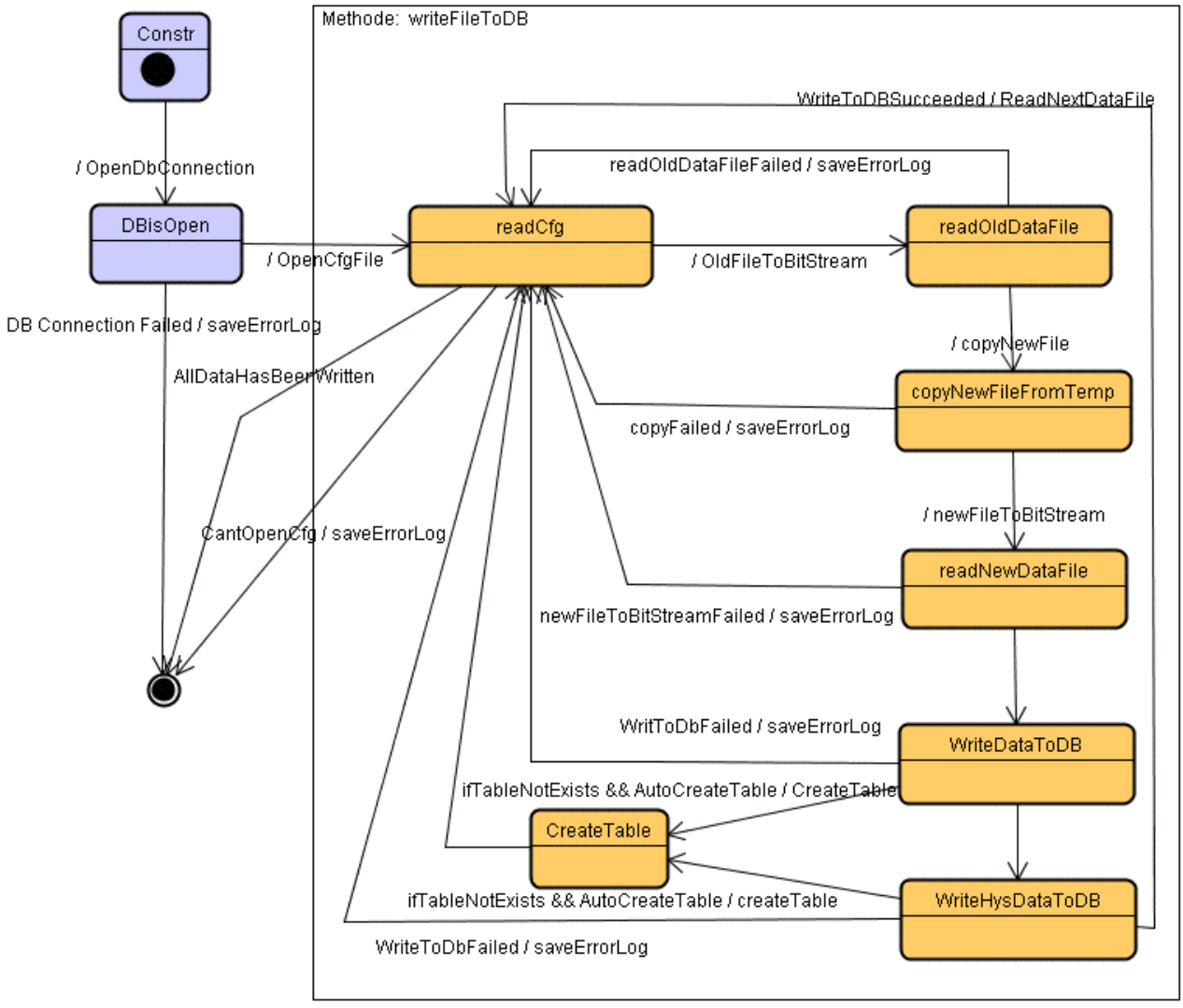


Abbildung 18: Zustandsdiagramm File Upload

Mit OldFileToBitStream() wird der Inhalt vom vorderen Zyklus aus dem File ausgelesen und in den Arbeitsspeicher kopiert, anschliessend wird das neu auf den Webserver hochgeladene File mit „copyNewFileFromTemp()“ vom Temporären Verzeichnis des Webservers ins Arbeitsverzeichnis kopiert und damit das alte überschrieben. Mit „readNewDataFile()“ wird das File ausgelesen. Die Methode writeDateToDB() vergleicht die Daten vom vorderen Kommunikationszyklus mit den aktuellen. Weichen

die alten Daten von den neuen ab, so wird mit dem folgenden SQL Befehl die Datenbank mit dem neuen Wert aufgefrischt:

```
INSERT INTO `table` SET `id`='id1',`field1`='data1',`field2`='data2' ON DUPLICATE KEY UPDATE `field1`='data1',`field2`='data2'
```

Ist kein Eintrag mit einem primary key Id1 vorhanden, wird ein neuer Datensatz mit den nach dem Schlüsselwort SET angegebenen Werten in die Tabelle eingefügt. Besteht bereits ein primary key mit dem Wert id1 so werden die alten Werte mit den neuen ersetzt. Zu beachten ist, dass die ON DUBLICATE KEY UPDATE Anweisung MySQL spezifisch ist, und nicht dem SQL-Standard entspricht, aber in diesem Fall äusserst praktisch ist.

Existiert die Tabelle table nicht und ist das Flag „$globCfg['createTables']“ im File receive_conf.php auf true gestellt, wird die Tabelle automatisch von der Methode writeDateToDB() erstellt.

Die Methode WriteHysDataToDB() schreibt die Daten in die Logging-Tabelle des mit fieldsLog angegebenen Feldes. Der Wert soll aber nur in die Tabelle geschrieben werden, wenn er sich vom zuletzt in die Tabelle eingetragenen Wert unterscheidet. Damit wird verhindert, dass die Tabelle mit unnötigen Redundanten Daten vollgeschrieben wird. Um nicht ein Abfrage-Statement und darauf wenn nötig ein Schreibstatement durchführen zu müssen, wird das folgende SQL-Statment verwendet:

```
INSERT INTO `hys_table` (`time`, `data`) SELECT actualTime, newValue FROM
DUAL WHERE NOT EXISTS (SELECT `time`,`data` FROM `hys_table` where
newValue = (SELECT `data` FROM `hys_table` ORDER BY `time` DESC LIMIT 0 ,
1));
```

Damit ist die Datenbank selbst in der Lage, zu bestimmen, ob der Wert in die Datenbank eingetragen werden muss oder nicht. Das INSERT-Statement wird nur ausgeführt, wenn das Resultat des Subselects nach WHERE NOT EXISTS kein Resultat liefert. Der Platzhalter actualTime beinhaltet die aktuelle Zeit, newValue ist der neu einzutragene Wert. Auch hier wird, wenn die Tabelle hys_table nicht existiert und das Flag $globCfg['createTables'] im File receive_conf.php auf true gestellt ist, die Tabelle automatisch erstellt.

Wie im Klassendiagramm in der nächsten Abbildung zu sehen ist, assozoziert die Klasse fileToDB die Klasse database. Sie kappselt die PHP-MySQL Datenbankfunktionen.

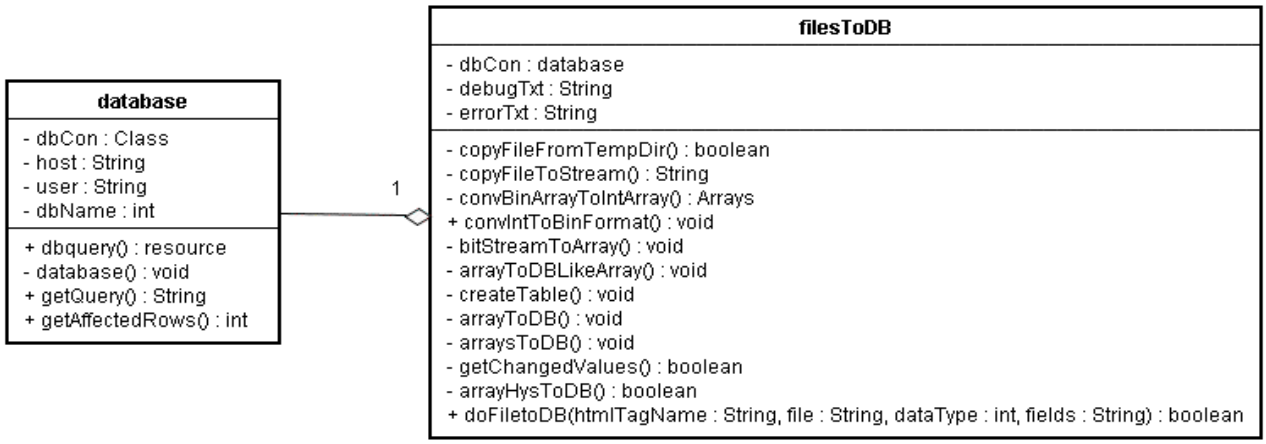

database
- dbCon : Class
- host : String
- user : String
- dbName : int
+ dbquery() : resource
- database() : void
+ getQuery() : String
+ getAffectedRows() : int
1
filesToDB
- dbCon : database
- debugTxt : String
- errorTxt : String
- copyFileFromTempDir() : boolean
- copyFileToStream() : String
- convBinArrayToIntArray() : Arrays
+ convIntToBinFormat() : void
- bitStreamToArray() : void
- arrayToDBLikeArray() : void
- createTable() : void
- arrayToDB() : void
- arraysToDB() : void
- getChangedValues() : boolean
- arrayHysToDB() : boolean
+ doFiletoDB(htmlTagName : String, file : String, dataType : int, fields : String) : boolean

Abbildung 19: Klassendiagramm Datenverarbeitung

6 Kommunikationssoftware Feldgerät

6.1 Design

Die Kommunikationssoftware auf dem Feldgerät stellt das Interface zwischen der Applikation und dem Webserver dar. Sie sendet in einstellbaren abständen die Aktuellen Daten zum Server und erhält ihrerseits die aktuellen Werte mit dem Response zurück. Der Server kann nicht von sich aus mit dem Client eine Kommunikation aufbauen. Dies ist in den allermeisten Anwendungen nicht nachteilig, beschränkt sich die Kommunikation vom Server zum Feldgerät auf Sollwertänderungen der Anlagen und ähnliches, während umgekehrt die Feldgeräte kontinuierlich ihr Prozessabbild auf den Server schreiben.

Im COB 0 läuft die Applikation der Steuerung. Diese ruft die Makros httpAddValuesToBufferR/F() mit den zu kommunizierenden Werten auf. Aufrufen ist nicht korrekt, eigentlich müsste man es eher beinhalten nennen, da Makros an der entsprechenden Stelle im Code expandiert werden. Makros haben eine Reihe Vorteile (siehe Kapitel Programmierung) gegenüber Funktionsaufrufen bei Saia-Steuerungen. Der Einfachheit halber spreche ich auch bei Makros von aufrufen. Immer im Wissen, dass Makros keine eigentlichen Funktionen sind.

Ein griffiges Interface in der Applikation zu haben, erschien mir wichtig. Daher gibt es lediglich zwei Routinen, an die die Applikationsroutinen ihre zu Kommunizierenden Werte übergeben können. Dies macht ein nachträgliches Hinzufügen von Werten leicht. Die httpAddValuesToBufferR/F() Makros schreiben die zu kommunizierenden Daten in denHTTP-Buffer. Gleichzeitig registrieren diese aber auch bei dessen erstem Aufruf die Adressen der Kommunikationsvariablen. Damit später bestimmt werden kann, an welche Adressen die empfangenen Daten geschrieben werden müssen. Die Routinen müssen bei jedem Zyklus aufgerufen werden. Diese schreiben die Werte in den Sendebuffer vom Http-Modul. Dieses wiederum ist dafür besorgt, die Daten an den Webserver auszuliefern:

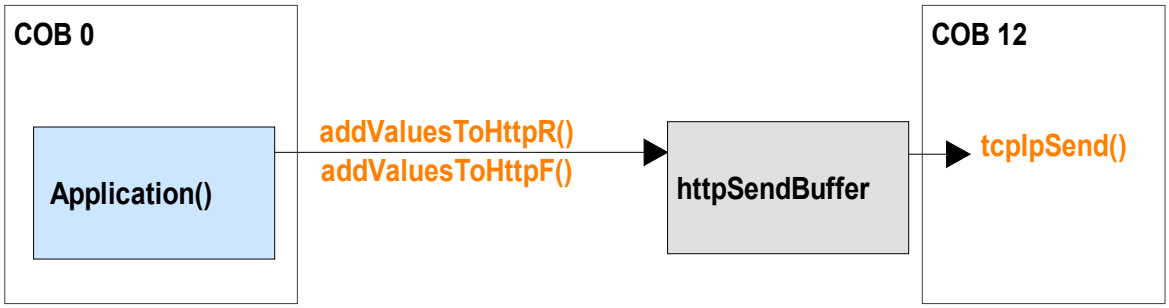


Abbildung 20: Daten senden

Nachfolgend ein Sequenzdiagramm vom Gesamtzusammenhang.

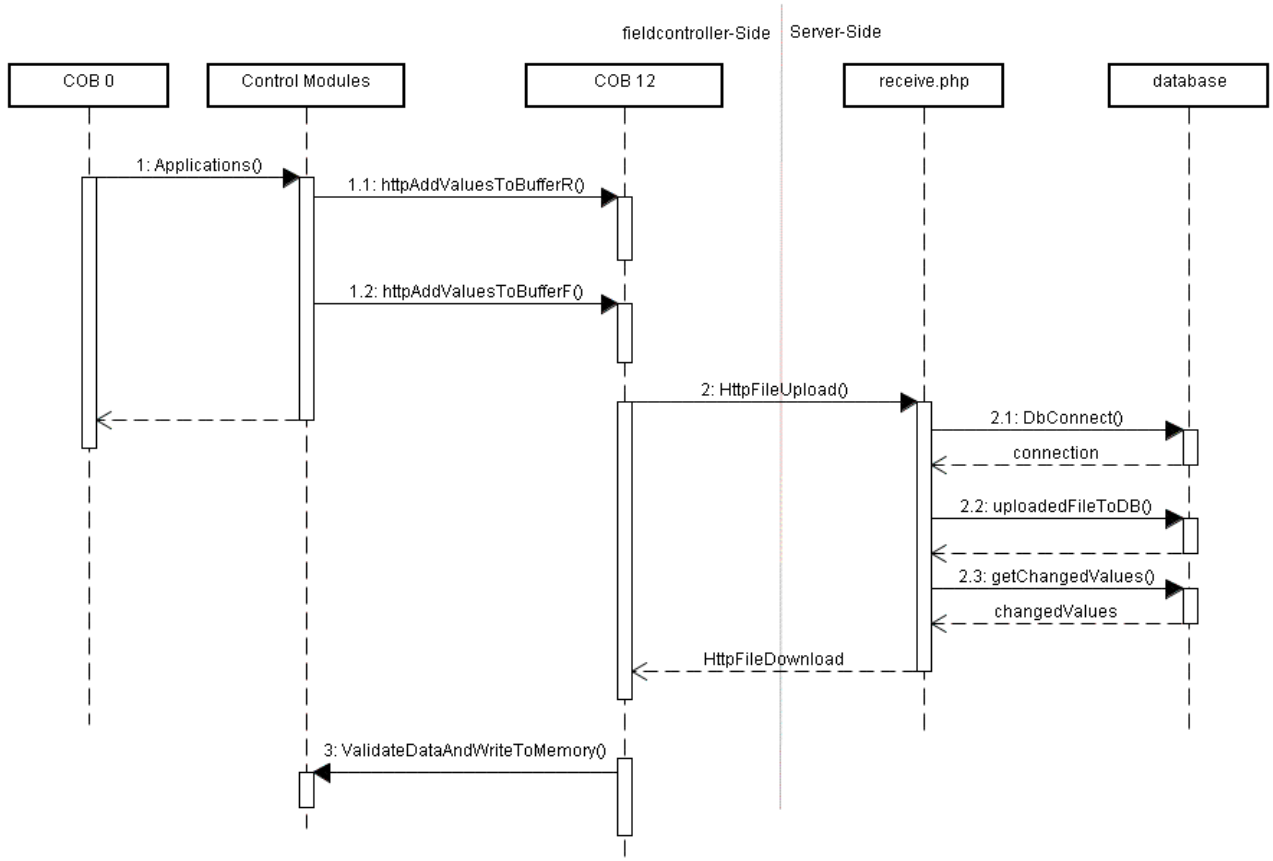


Abbildung 21: Sequenzdiagramm Gesamtablauf

Zuerst werden einige Initialisierungschritte gemacht. Im State-InitTCPIP wird die Saia-TCP/IP-Schnittstelle initialisiert. Für jedes zu sendenden File muss die die Grösse ermittelt werden, damit in den nächsten Schritten der dafür nötige Platz im Sendebuffer reserviert werden kann. Die zu sendenden Bytes pro File werden beim ersten Aufruf von httpAddValuesToBuffer() ermittelt.

Der Sendebuffer entspricht Byte für Byte dem HTTP-Request, der später über die TCP/IP-Schnittstellen gesendet wird. Deshalb muss der Http-Sendbuffer mit denn nötigen HTTP-String-Grundgerüst formatiert werden. Der Aufbau des Buffers ist in Abbildung 23 gezeigt und entspricht der Beschreibung im Kapitel 4,. HTTP-Request zum Server S.26. Das Makro httpAddCharsToBuffer() schreibt zuerst den HTTP Header in den Sendebuffer. Für jedes zu kommunizierende File muss ein Filehead erstellt werden, darauf folgen die binären Daten. Diese können erst nach erfolgter Formatierung eingefügt werden. Gleichzeitig schreibt es die Position der File-anfänge im Sendebuffer in ein Array, damit nach erfolgter Formatierung die Daten auch an die richtige Position geschrieben werden.

Abgeschlossen wird ein File mit einem speziellen Trennstring. Der State calculateAndWriteContLength berechnet die Länge der Content-Daten und schreibt diese in das Feld Content-Length im Sendebuffer. Des weiteren wird berechnet, wieviele TCP/IP-Pakete für ein HTTP-Request gesendet werden müssen. Danach ist die Initialisie-

rung des Sendebuffers abgeschlossen und das Programm kann in den Sendemodus übergehen.

Wenn der Timer (im State CheckCommIntervall) abgelaufen ist, wird zunächst eine TCP/IP-Verbindung zum Webserver aufgebaut. Dann wird Paket um Paket aus dem Sendebuffer ausgelesen und an den Webserver gesendet.

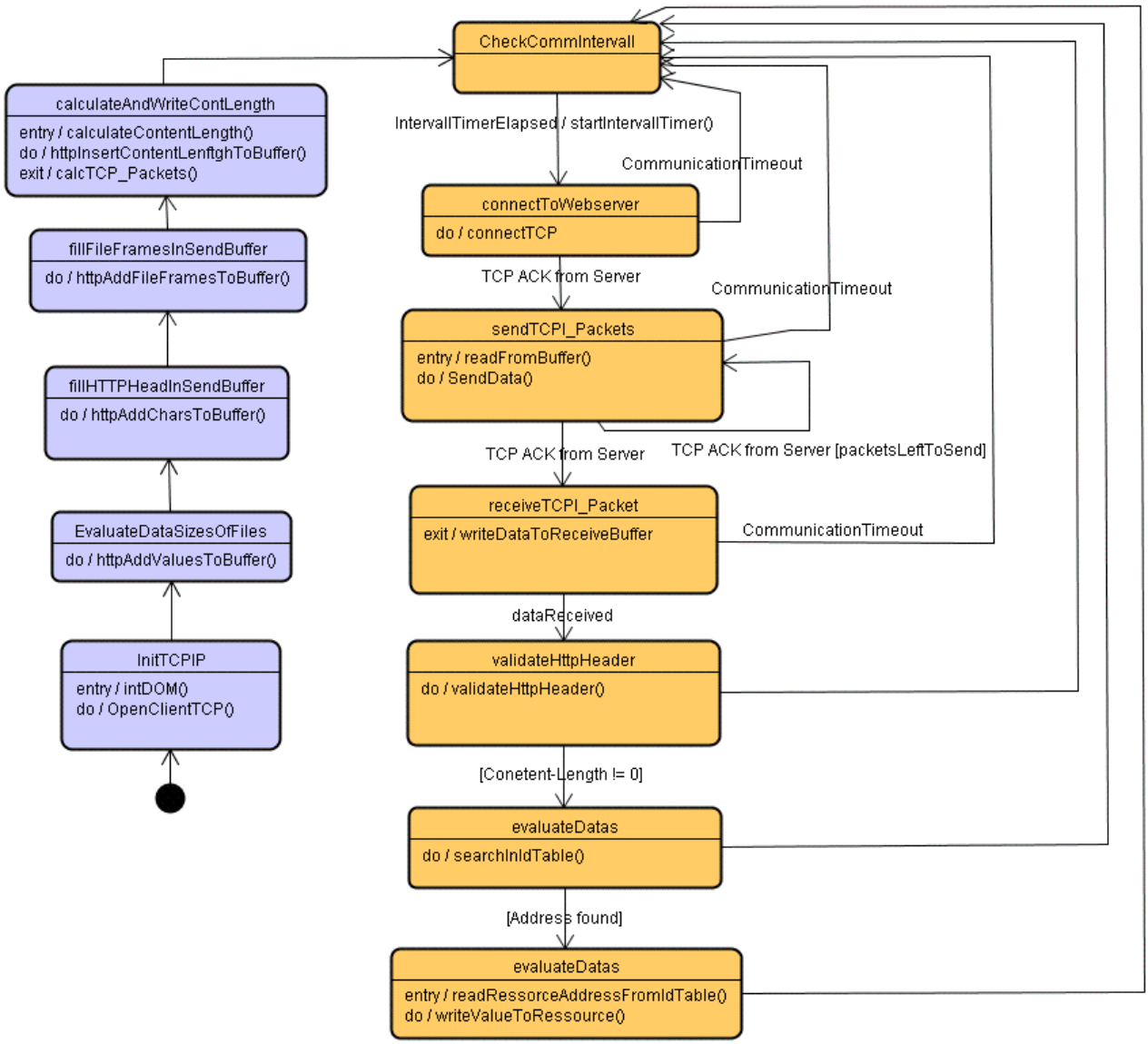


Abbildung 22: State Event Kommunikation Feldgerät

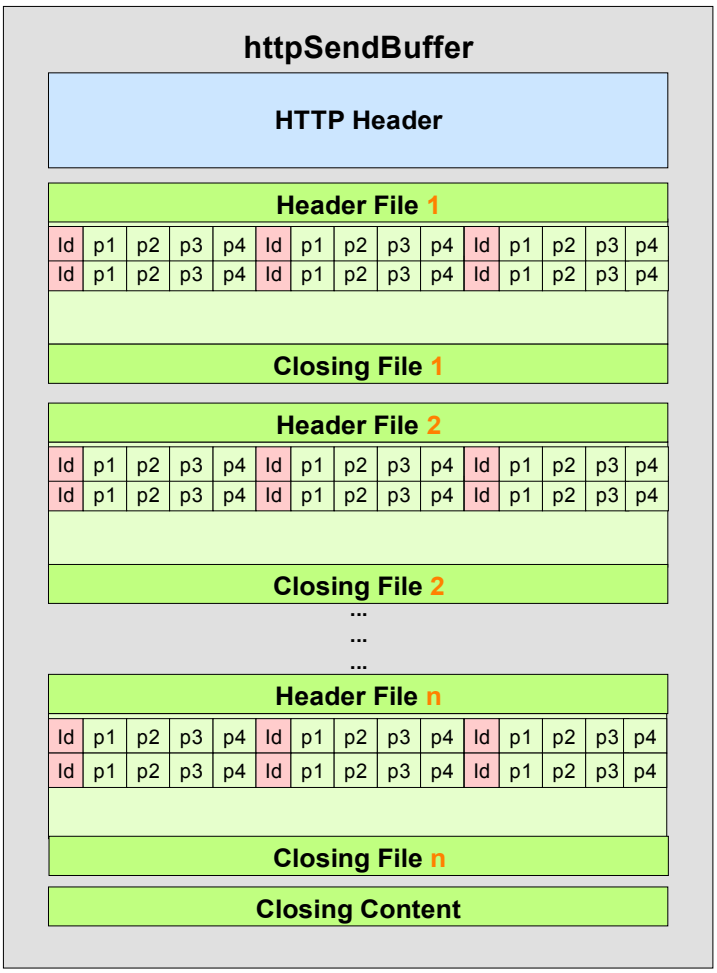

httpSendBuffer
HTTP Header
Header File 1
Id p1 p2 p3 p4 Id p1 p2 p3 p4 Id p1 p2 p3 p4
Id p1 p2 p3 p4 Id p1 p2 p3 p4 Id p1 p2 p3 p4
Closing File 1
Header File 2
Id p1 p2 p3 p4 Id p1 p2 p3 p4 Id p1 p2 p3 p4
Id p1 p2 p3 p4 Id p1 p2 p3 p4 Id p1 p2 p3 p4
Closing File 2
...
...
...
Header File n
Id p1 p2 p3 p4 Id p1 p2 p3 p4 Id p1 p2 p3 p4
Id p1 p2 p3 p4 Id p1 p2 p3 p4 Id p1 p2 p3 p4
Closing File n
Closing Content

Abbildung 23: Aufbau Sendbuffer

Der Aufbau des Http-Requests lässt sich eins zu eins im Sendebuffer abbilden, so dass die TCP/IP-Routine nichts anderes zu machen braucht, als den Buffer zu senden. In Abbildung 23 ist die Gliederung vom Request bzw. Sendebuffer gezeigt. Er besteht aus dem HTTP-Header, gefolgt vom „Header File 1“, vom binären Inhalt, wie im Kapitel 4 HTTP-Request zum Server beschrieben gefolgt vom „Closing File 1“ Auf diese Weise können mehre Files aneinandergereiht werden. Zum Schluss muss der content noch abgeschlossen werden.

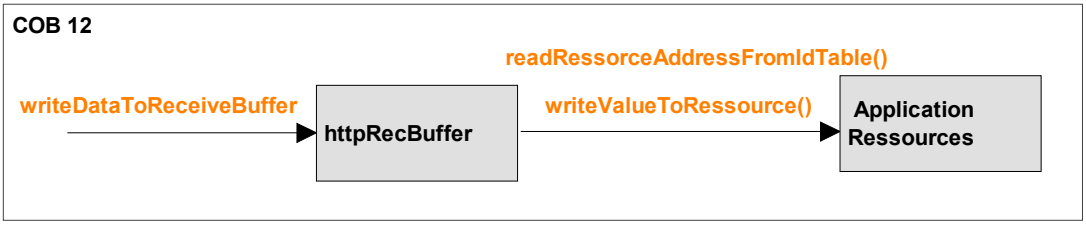


Abbildung 24: Empfangen, HTTP-Response des Webservers

Die Verarbeitung der empfangenen Daten lässt sich in folgende Hauptteile gliedern: Die HTTP-Daten werden vom receiveData() Routine empfangen und anschliessend in den HTTP-Empfangsbuffer geschrieben. Die Daten werden vom validateHttpHeader() auf Datentyp, Länge etc. geprüft. Die Validierung des HTTP-Headers sieht folgende Überprüfungen vor:

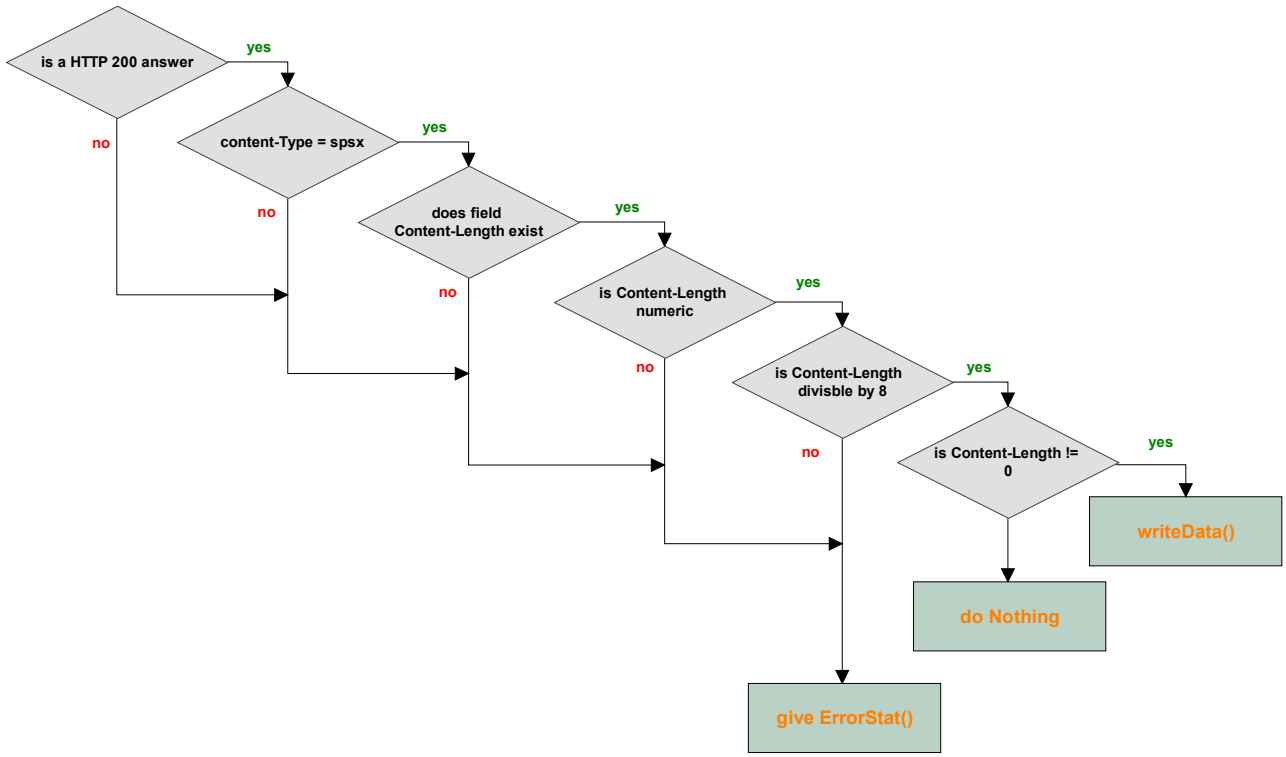


Abbildung 25: Validierung Server Response

Zuerst wird überprüft, ob der Server eine HTTP 200 OK gesendet hat. Danach muss sichergestellt werden das der Application-Type = spsx ist. Spsx ist ein individuell festgelegter Content-Type. Damit wird sichergestellt, dass der Inhalt im Body auch dem erwarteten Format entspricht. Ansonsten besteht die Gefahr aus Versehen eine Textmessage, oder eine Postkarte in den Speicher des Feldgerätes einzulesen. Als nächstes wird überprüft, ob im Header ein Feld „Content-Length„ vorhanden ist. Wurde das Feld gefunden, so wird als nächstes überprüft, ob die Länge numerisch ist. Das heisst, die Längenangabe muss aus ASCII Zeichen zwischen 0-9 bestehen. Ein Schlüssel/Wert-Paar im Content besteht zusammen aus 8 Byte, daher wird als nächstes geprüft ob die Content-Länge durch 8 teilbar ist. Ist die Länge zudem ungleich null, werden die gesendeten Daten eingelesen und interpretiert.

Der empfangene Inhalt muss nun weiterverarbeitet werden. Die Daten liegen wie folgt organisiert als Bitstream im Empfangsbuffer:

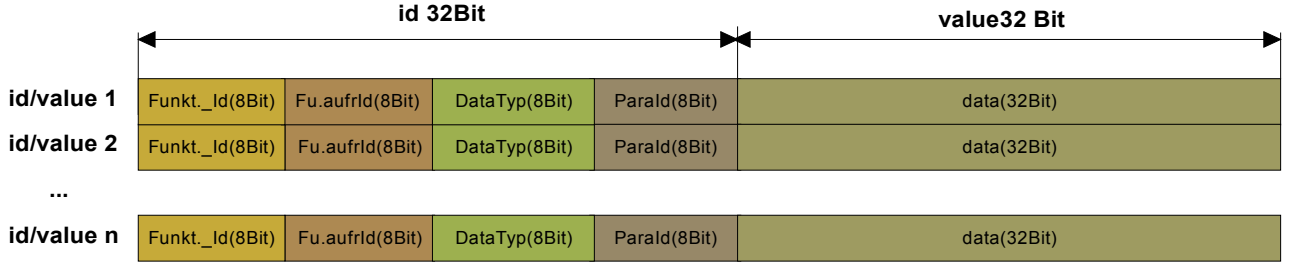


Abbildung 26: Datenorganisation im Empfangsbuffer

Immer 64 Bit (8Byte) beinhalten ein Id mit dem ihr zugeordneten Wert (value). Im Gegensatz zum Request sendet der Server nur geänderte Daten an den Client. Deshalb muss der Server eine Id zu jedem Wert(value) mitsenden. Die Id Setzt sich aus der Funktions-Id, der Funktionsaufruf-Id, dem Datentyp und der Parameter Id zusammen, wie dies in Kapitel 4 HTTP-Request zum Server beschrieben wurde. Die Länge der Teil-Ids ist jeweils 8 Bit. Dies bedeutet eine Einschränkung auf maximal 256 Funktionen, maximal 256 Aufrufe einer Funktion, maximal 256 Datentypen und maximal 256 Parameter pro Funktion. Byte-Operationen werden vom Controller effizienter ausgeführt, deshalb die Wahl auf gleichgrosse 8-Bit Einheiten.

Der Client führt einen Address Lookup Table welcher aus zwei Arrays besteht. Eines mit den Ids und eines mit der dazugehörenden Speicheradressen sowie deren Datentyp. binarySearch() sucht im „ID-Array“ nach der vom Webserver gesendeten identischen Id.

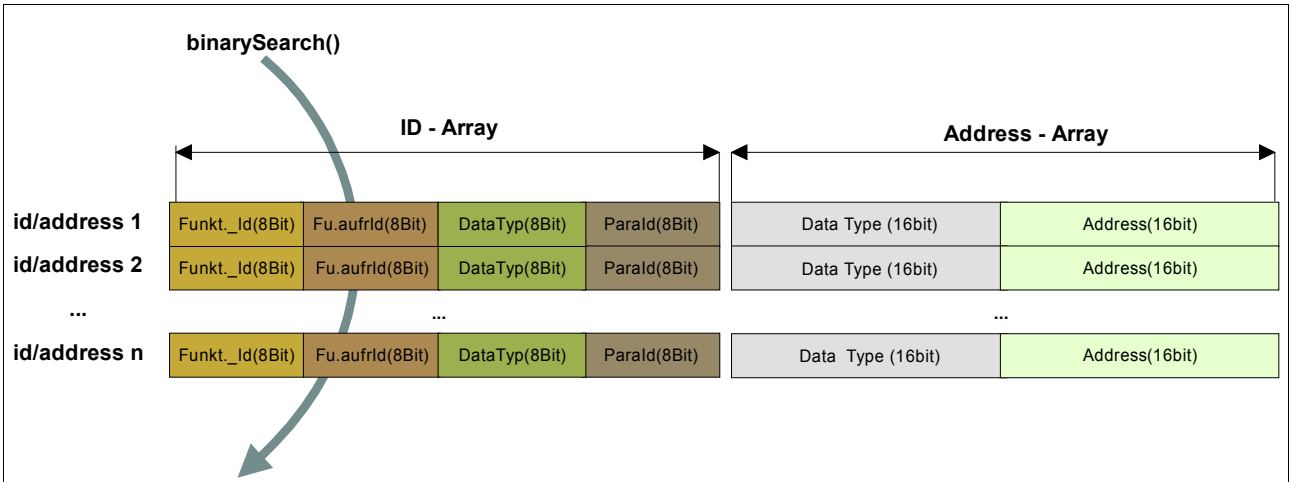


Abbildung 27: Address lookup t able

Da maximal 16384 Register und maximal 8192 Flags anzusprechen sind, sind 16 Bit für den Adressraum mehr als ausreichend. Würde der Adressraum nicht ausreichen kann „Data Type“ und „Address“ in 2 Arrays aufgeteilt werden. Damit könnten 2^32 Adressen und 2^32 Datentypen angesprochen werden.

Wird ein ID/Wert-Paar vom Server empfangen, muss das ID-Array nach der übereinstimmenden Id durchsucht werden. Sollen hunderte von Variablen kommuniziert werden (und damit ist zu rechnen) beinhaltet doch die vorliegende, relativ kleine Musteranlage alleine über 500 kommunizierbare Register und noch mehr Flags, würde eine Lineare Suche viel zu viel Rechenzeit benötigen. Daher muss eine andere Lösung gefunden werden. Die Daten sollten geordnet abgelegt werden, zum Beispiel in einer Baumstruktur, ein Heap wäre dazu geeignet, kann dieser doch in einem Array gespeichert werden und belegt nicht, wie in einer Baumstruktur zusätzlich Speicher für die Knotenadressen. In einem Heap können mit relativ kleinem Aufwand (proportional zur Höhe des Baumes oder log2 zur Anzahl Elemente) Elemente ersetzt und gelöscht werden. Dies ist im konkreten Fall nicht nötig, die Id-Liste muss beim Aufstarten erfasst werden und bleibt danach statisch. Also würde auch eine Sortierte Liste reichen. Der Suchaufwand in einem Heap oder in einer sortierten Liste ist bedeutend kleiner als bei einer linearen Suche. Vorausgesetzt man durchsucht die sortierte Lis

te mit einem „Divide and conquer“-Algorithmus, auch binäre Suche (oder Englisch binary search) genannt:

$$Omax = log2(n+1)$$

Die maximale Anzahl Operationen (Omax) um im „Worst Case“ ein Element in einer sortierten Liste zu finden, steigt nur logarithmisch zur Anzahl Elemente (n), wie dies in der obigen Formel zu erkennen ist. So sind für 1023 Werte lediglich maximal zehn Operationen nötig um den gesuchten Wert zu finden. Für die doppelte Anzahl Elemente wäre demnach nur eine Operation mehr nötig um das gesuchte Element zu finden. Wurde eine entsprechende ID im ID-Array gefunden so gibt die binarySearch() Routine den Index zurück. Mit dem Index kann nun im Address-Array die entsprechende Adresse und der Datentyp ausgelesen werden. WriteValueToRessource() überschreibt nun die Ressource mit dem empfangenen Wert (value).

6.2 Umsetzung

Die Aufgabe wird in drei Teile aufgeteilt, in eine Klasse-COM02, in ein Klasse-Http und eine Klasse sortedList. Die Programmierung habe ich an die Objekt Orientierte Welt angelehnt. Einige Ideen und Prinzipien können von der objektorientierten Welt in die konventionelle prozedurale Welt übertragen werden. Natürlich, einige wichtige Elemente wie Vererbung, Polymorphie etc. lassen sich schwerlich umsetzten.

Schaut man die Datenorganisation eines Objektes an, so besteht ein Objekt aus einer Gruppe von Attributen und den dazugehörenden Funktionen. Diese Art von Kohäsion lässt sich auch in einer nicht objektorientierten Programmiersprache abbilden, um das Konzept verständlich zu machen, nachfolgend ein Beispiel in C. Die Attribute können in eine Struktur gepackt werden, die dazugehörenden Methoden (Funktionen) können syntaktisch aneinander gebunden werden.

```
typedef struct
{
      char *            recBufferTcp;
      char *            searchedStrHttpOk;
      char *            searchedStrContentLentgh;
      unsigned int      httpRecContentLength_Dat;
      BOOL              httpRecContentLengthDataNotOk;
} COM02 = { „HTTP 200 OK\r\n“,
            „Content-Length: “,
            0,
            0};
COM02 myCom02;

COM02_SearchString(COM02* this,char * searchedString,int searchedStringLentgh)
{
  int i;
  int j;
  i = 0;
  while (((*(this->recBufferTcp)) + i) != '\0')
  {
      j = 0;
      while (((*(this->recBufferTcp)) + i) == ((*(this->searchedString))+j))
      {
         if (j =< (searchedStringLentgh-1))
         {
            this->searchedStrHttpOk = true;
            return i;
         }
         j++;
      }
      i++;
  }
}

#include <stdio.h>
int main( int argc, const char* argv[] )
{
      com02SearchString(myCom02,myCom02->searchedStrHttpOk,strlen(myCom02
                  ->searchedStrHttpOk));
}
```

Kurz gesagt handelt es sich hier um eine Ansammlung von Funktionen mit gleich beginnendem Namen, welche alle die gleiche Struktur als Parameter aufnehmen.

Das obige C-Beispiel zeigt die Klasse COM02. Sie besteht aus einigen Attributen und der Methode „com02SearchString“. Sie sucht mit einer linearen Suche in einem String nach einem anderen String. Alle Methoden der Klasse beginnen mit „COM02_“. Der „Klasse“ muss natürlich eine Referenz der ihr zugehörenden Attribute mitgegeben werden. Diese wird den Methoden mit einem Zeiger auf die Struktur des jeweiligen Objektes mitgegeben. Innerhalb der Methoden können die Attribute des Objektes mit dem „this“-Zeiger angesprochen werden.

Ähnliche Konstrukte kann man auch mit Saia-AWL und mithilfe der Makros und Variablen-Namen erzeugen. Namensräume können mit einem Punkt oder mit einer $GOUP-Klausel erzeugt werden. Die Struktur (struct) der „http“-Klasse würde dann wie folgt aussehen (siehe auch S. Kapitel):

```
$GROUP
http.n1 kSendReqLength     EQU    http.n1.kIdVEN03+1
Flag.rCount          EQU    R [http.n1.kSendReqLength]
Register.rCount      EQU    R [http.n1.kSendReqLength]
Flag.rPosition       EQU    R [http.n1.kSendReqLength]
Register.rPosition EQU    R [http.n1.kSendReqLength]
rFillPointer         EQU    R
rObjectOffsF         EQU    R
rObjectOffsR         EQU    R
rBitRegister         EQU    R
rFillRegister        EQU    R
rOffsetCounterR      EQU    R
rOffsetCounterF      EQU    R
$ENDGROUP
```

Und einen Methodenaufruf der Klasse HTTP:

```
 http.addCharsToBuffer(http.n1,0,'P','O','S','T',' ','/','p','v','/','r','e','c'
,'.','p','h','p',' ','H','T','T','P','/','1','.','1','<CR>','<LF>')
```

Eine solches Design erhöht die Wiederverwendbarkeit und erleichtert die Lesbarkeit des Codes beträchtlich. So könnte die Klasse http jetzt beliebig oft „instanziert“ werden. Vermutlich liesse sich die Programmierung noch weiter in die Objektorientierte Richtung treiben. So könnte eventuell das Factory–Pattern verwendet werden, um die nötige Struktur zusammen mit den Routinen (Makros) mit demselben Namensraum zu erzeugen. Leider kann ich hier den objektorientierten Ansatz aus Zeitgründen nicht noch weiterausbauen.

In Abbildung 28 ist ein Klassendiagramm aufgeführt. Die Struktur besteht aus der Hauptklasse COM02 welche auch die Service-Schnittstelle beinhaltet.

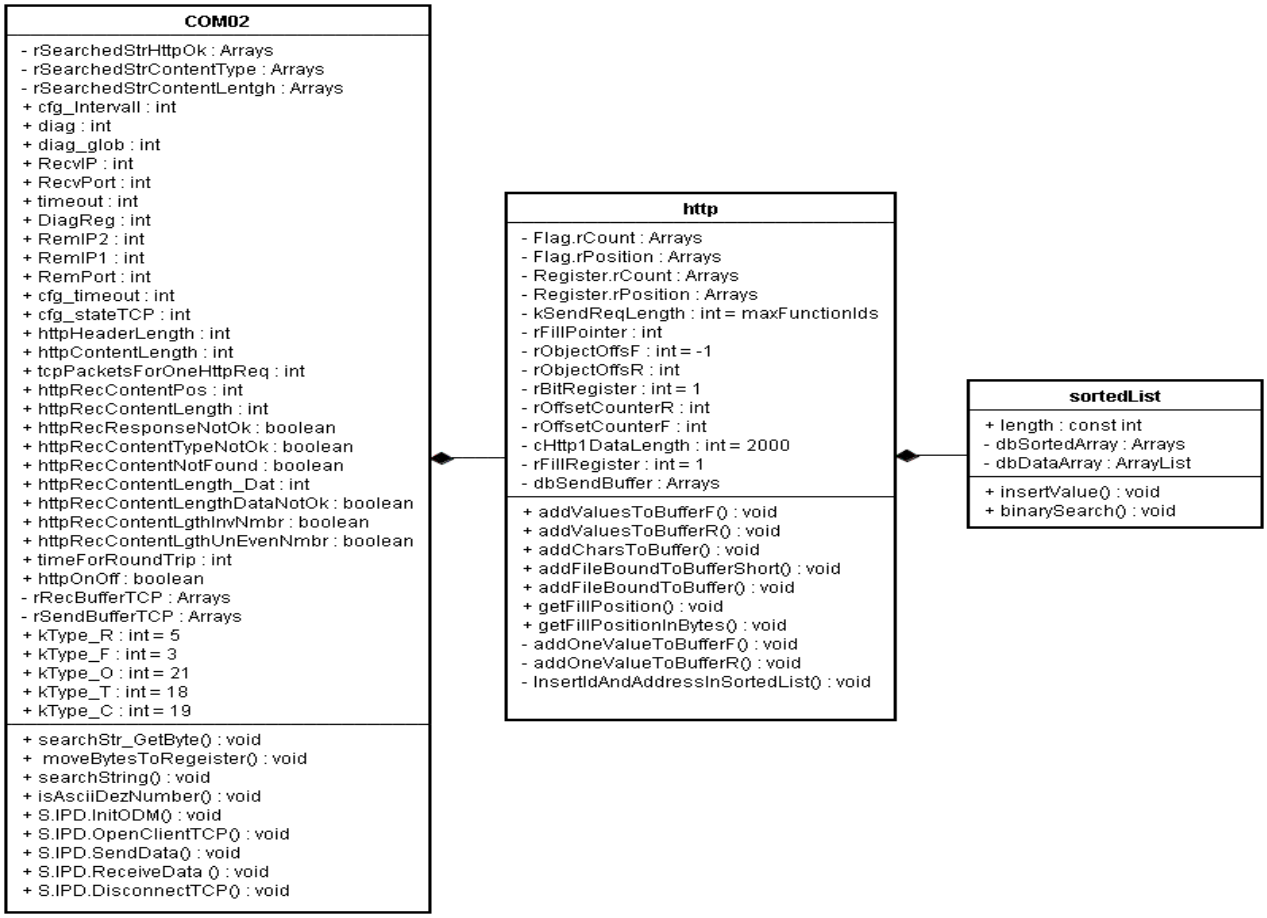

COM02
- rSearchedStrHttpOk : Arrays
- rSearchedStrContentType : Arrays
- rSearchedStrContentLentgh : Arrays
+ cfg_Intervall : int
+ diag : int
+ diag_glob : int
+ RecvIP : int
+ RecvPort : int
+ timeout : int
+ DiagReg : int
+ RemIP2 : int
+ RemIP1 : int
+ RemPort : int
+ cfg_timeout : int
+ cfg_stateTCP : int
+ httpHeaderLength : int
+ httpContentLength : int
+ tcpPacketsForOneHttpReq : int
+ httpRecContentPos : int
+ httpRecContentLength : int
+ httpRecResponseNotOk : boolean
+ httpRecContentTypeNotOk : boolean
+ httpRecContentNotFound : boolean
+ httpRecContentLength_Dat : int
+ httpRecContentLengthDataNotOk : boolean
+ httpRecContentLgthInvNmbr : boolean
+ httpRecContentLgthUnEvenNmbr : boolean
+ timeForRoundTrip : int
+ httpOnOff : boolean
- rRecBufferTCP : Arrays
- rSendBufferTCP : Arrays
+ kType_R : int = 5
+ kType_F : int = 3
+ kType_O : int = 21
+ kType_T : int = 18
+ kType_C : int = 19
+ searchStr_GetByte() : void
+ moveBytesToRegeister() : void
+ searchString() : void
+ isAsciiDezNumber() : void
+ S.IPD.InitODM() : void
+ S.IPD.OpenClientTCP() : void
+ S.IPD.SendData() : void
+ S.IPD.ReceiveData () : void
+ S.IPD.DisconnectTCP() : void
http
- Flag.rCount : Arrays
- Flag.rPosition : Arrays
- Register.rCount : Arrays
- Register.rPosition : Arrays
- kSendReqLength : int = maxFunctionIds
- rFillPointer : int
- rObjectOffsF : int = -1
- rObjectOffsR : int
- rBitRegister : int = 1
- rOffsetCounterR : int
- rOffsetCounterF : int
- cHttp1DataLength : int = 2000
- rFillRegister : int = 1
- dbSendBuffer : Arrays
+ addValuesToBufferF() : void
+ addValuesToBufferR() : void
+ addCharsToBuffer() : void
+ addFileBoundToBufferShort() : void
+ addFileBoundToBuffer() : void
+ getFillPosition() : void
+ getFillPositionInBytes() : void
- addOneValueToBufferF() : void
- addOneValueToBufferR() : void
- InsertIdAndAddressInSortedList() : void
sortedList
+ length : const int
- dbSortedArray : Arrays
- dbDataArray : ArrayList
+ insertValue() : void
+ binarySearch() : void

Abbildung 28: Klassendiagramm Kommunikation Feldebene

Klassse COM02

Attribute: Die Attribute sind ausschliesslich Kommunikationsvariablen, welche von der Serviceschnittstelle von ProMosNT bedient werden.

Methoden:

InitODM()
Initialisiert den-Open-Data-Mode (Saia API)

OpenClientTCP()
Öffnet einen Client-TCP-Kanal im IP Open Data Mode (Saia API).

ConnectTCP()
Verbindet einen TCP-Kanal im IP Open Data Mode mit einem Server-TCP-Kanal.(Saia API)

SendData()
Sendet Daten über einen Kanal im IP Open Data Mode (Saia API).

ReceiveData()
Empfängt Daten über einen Kanal im IP Open Data Mode (Saia API).

DisconnectTCP()

Trennt einen TCP-Kanal im IP Open Data Mode (Saia API).

Close() Schliesst einen Kanal im IP Open Data Mode (Saia API).

SerchStr_GetByte() Extrahiert ein Byte aus einem Register.

SearchString() Sucht linear in einer Zeichenkette nach dem Vorkommen einer anderen Zeichenkette. Bei Erfolg wird die Endposition der gefundenen Zeichenkette zurückgegeben. Wurde die Zeichenkette nicht gefunden, wird -1 zurückgegeben. Müssten längere Zeichenketten durchsucht werden, so könnte zum Beispiel ein Boyer-Moore Algorithmus verwendet werden. Dieser arbeitet effizienter als die lineare Suche.

IsAsciiDezNumber() Gibt TRUE zurück, wenn der Parameter einen Wert zwischen 48 – 57(ASCII 0-9) hat.

Klasse HTTP

Attribute:

Flag.rCount ist ein Integer-Array von der Länge kSendReqLength. Also die maximale Anzahl Funktions-Ids. Es beinhaltet die Anzahl Flags, die für jede Funktion (Methode) Kommuniziert werden müssen.

Register.rCount Ist ein Integer-Array von der Länge kSendReqLength. Also die maximale Anzahl Funktions Ids. Es beinhaltet die Anzahl Register die für jede Funktion (Methode) Kommuniziert werden müssen.

Flag.rPosition Ist ein Integer Array von der Länge kSendReqLength. Es beinhaltet für jede Funktion die Startposition im Sendebuffer. Damit zur Laufzeit jede Funktion die Register- daten an den richtige Ort schreiben kann.

Register.rPosition Ist ein Integer array von der Länge kSendReqLength. Es beinhaltet für jede Funktion die Startposition im Sendebuffer. Damit zur Laufzeit jede Funktion die Flag-Daten an den richtige Ort schreiben kann.

rFillPointer wird zum befüllen des Sendebuffers mit dem Header und den Fileheader und Abgrenzungsstrings benötigt. Speichert im wesentlichen die aktuelle Position des zuletzt geschriebenen Zeichens.

rObjectOffsR wird zum Füllen des Sendebuffers benötigt.

RobjectOffsF wird zum Füllen des Sendebuffers benötigt.

RbitRegister Bits müssen als 32 Bit Werte in den Sendebuffer geschrieben werden. Hierfür wird jedes Bit zuerst im rBitRegister gespeichert, ist dieses voll oder möchte eine neue Funktion in den Sendebuffer schreiben, so wird das rBitRegister als ganzes in den Sendebuffer kopiert.

rOffsetCounterR Wird benötigt, um Register in den Sendebuffer zu schreiben. Zeigt immer auf die Position der nächsten überschreibbaren Register-Position im Sendebuffer.

ROffsetCounterF Wird benötigt um Flags in den Sendebuffer zu schreiben. Zeigt immer auf die Position der nächsten überschreibbaren Flag-Position im Sendebuffer.

chHttp1Length Länge des Sendebuffers in Anzahl Register. Ist mit vier zu multiplizieren um Bytes zu erhalten.

rFillRegister Wird zum fFüllen des Sendebuffers mit ASCII-Zeichen (8-Bit Werten)benötigt. In Datenblöcke können nur Register (32-Bit-Werte) eingefügt werden. Daher wird Byte für Byte zuerst in das rfillRegister geschrieben. Ist es voll, kann es in den Sendebuffer kopiert werden.

DbSendBuffer Ist ein Datenblock. Ein Datenblock ist ein Array von 32-Bit-Werten.

Methoden:

addValuesToBufferF(objName,modId,id,v1..40)
kopiert Bit Werte in den Sendebuffer.
objName: Objektname.
modId: Die Funktions Id.
Id: Die Funktionsaufruf Id
v1:v40: Es können wahlweise 1..40 Flags als Argumente mitgegeben werden.

addValuesToBufferR(objName,modId,id,v1..40)
kopiert Register Werte in den Sendebuffer.
objName: Objektname.
modId: Die Funktions Id.
Id: Die Funktionsaufruf Id
v1..v40: Es können wahlweise 1..40 Register als Argumente mitgegeben werden.

addCharsToBuffer(objName,pos,v1..40)
fügt Zeichenketten in den Sendebuffer ein.
objName: Objektname.
pos: Die position an der der String in den Sendebuffer eingefügt werden soll Das Attribut rFillPointer wird auf pos gesetzt und mit jedem eingefügtem Zeichen inkremetiert. pos kann auch weggelassen werden, dann wird rFillPointer nur inkrementiert.
V1..v40: Es können wahlweise 1..40 Zeichen mitgegeben werden.

addFileBoundToBufferShort(objName,pos)
fügt ein File-Trennstring (für den Header) in den Sendebuffer ein.
objName: Objektname.
pos: siehe Methode
addCharsToBuffer()

addFileBoundToBuffer(objName,pos)
fügt ein File-Trennstring in den Sendebuffer ein.
objName: Objektname.
pos: siehe Methode addCharsToBuffer()

getFillPosition(objName,rOut)
gibt den Werte von rFillPointer zurück
objName: Objektname.
rOut: Rückgabewert (Inhalt rFillPointer).

getFillPositionInBytes(objName,rOut) gibt den Wert die aktuell Position von rFillPointer in Bytes zurück. Falls noch daten im rFillRegister sind, werden diese auch mitgezählt. objName: Objektname. rOut: rückgabewert.

addOneValueToBufferF(objName,value) kopiert den Inhalt eines Flags in den Sendebuffer. objName: Objektname.
value: fakultativ Paramater. Das Makro wird nicht expandiert wenn kein value Para-

meter mitgegeben wird.

addOneValueToBufferR(objName,value)
kopiert den Inhalt eines Registers in den Sendebuffer.
objName: Objektname.
value: fakultativ Paramater. Das Makro wird nicht expandiert wenn kein value Parameter mitgegeben wird.

InsertIdAndAddressInSortedList(objName,idMod,idDataSet,dataType,idParamter,rAddr)
erzeugt aus idMod, idDataSet,dataType und idParameter eine Id (Siehe auch Abbildung 27 Address lookup t able) und fügt diese in das das sortierte Id-Array ein. Die Adresse und der Ressourcetype von rAddr wird parallel dazu in das Address-Array eingefügt.
objName: Objektname.
IdMod: Modul oder Funktions-Id
idDataset: Datensatz Id oder Funktionsaufruf-Id
dataType: Kommunikationsdatetyp
idParamter: Parameter
Id rAddr: Ressource die registriert werden soll.

Klasse sortedList

Attribute

dbSortedArray ist ein Datenblock, der die sortierte Liste „ID-Array“ beinhaltet.

dbDataArray ist ebenfalls ein Datenblock der, alle Ressourcen „Address-Array“ beinhaltet. Die Liste wird parallel mit „dbSortedArray“ mitgeführt.

Methoden InsertValue(objName,rNewElement,dataArray1,dataVal1)
fügt einen Wert in dbSortedArray ein. Das neue Element wird solange mit dem nächst niedrigen Element getauscht, bis der Wert des vorangestellten Elementes kleiner oder gleich gross ist wie das einzufügende. Ist die endgültige Position des Elementes gefunden, so wird an derselben Stelle im dataArray1 der Parameter dataVal1 eingefügt.
objName: Objektname.
rNewElement: Neues in „dbSortedArray“ einzufügende Element
dataArray1: Datenarray hier wird dataVal1 an der gleichen Position wie rNewElement eingefügt.
DataVal1: einzufügender neuer Wert.

BinarySearch(yy,xx,array,length,ff_ii,DESC_ASC)
Sucht nach der Divide & Conquer Methode in einem Datenblock (Array) nach dem identischen Werte. Ist ein identischer Wert gefunden. wird der Index in yy ausgegeben, und der ACC ist =H gesetzt. Ist kein identischer Wert zu finden , wird die linke Begrenzung von den beiden Begrenzungszeiger zurückgegeben und der ACC = L gesetzt.
xx: gesuchter wert
yy: position
ff_ii: float oder integer
array 0=integer, 1=float
array: datenblock

length: länge des datenblocks
DESC_ASC: ASC = 0, liste ist aufsteigend sortiert DESC_ASC: ASC = 1, liste ist absteigend sortiert.

In Abbildung 29 ist der Gesamtablauf des Kommunikationsprozesses abgebildet. In Init1() wird die Kommunikaionschnittstelle initialisiert. Sie wird einmalig beim Start aufgerufen. Danach werden beim ersten Aufruf von addValuesToBufferR() und addValuesToBufferF() die Anzahl Flags und Register gezählt, gleichzeitig werden die Adressen der Ressourcen in die sortierte Liste geschrieben.

Da nun die Menge der zu Kommunizierenden Ressourcen bekannt ist, kann der HTTP Request in der Routine init2() in den Sendebuffer geschrieben werden. Jetzt ist die Initialierung abgeschlossen und es kann weiter in den Sende/Empfangsmodus im Kommunikationsloop übergegangen werden. Die Aplikation läuft im COB 0 während die Kommunikation im COB 12 läuft. COB 12 gibt seine Rechenzeit jeweils mit einem NCOB an COB 0 ab wenn er auf ein ACK vom Server wartet.

Zyklusmessungen im COB 0 zeigen dass die Prozessorlast durch die Kommunikation kaum gestiegen ist. Sie liegt mit oder ohne COB 12 bei ca. 6 milisekunden.

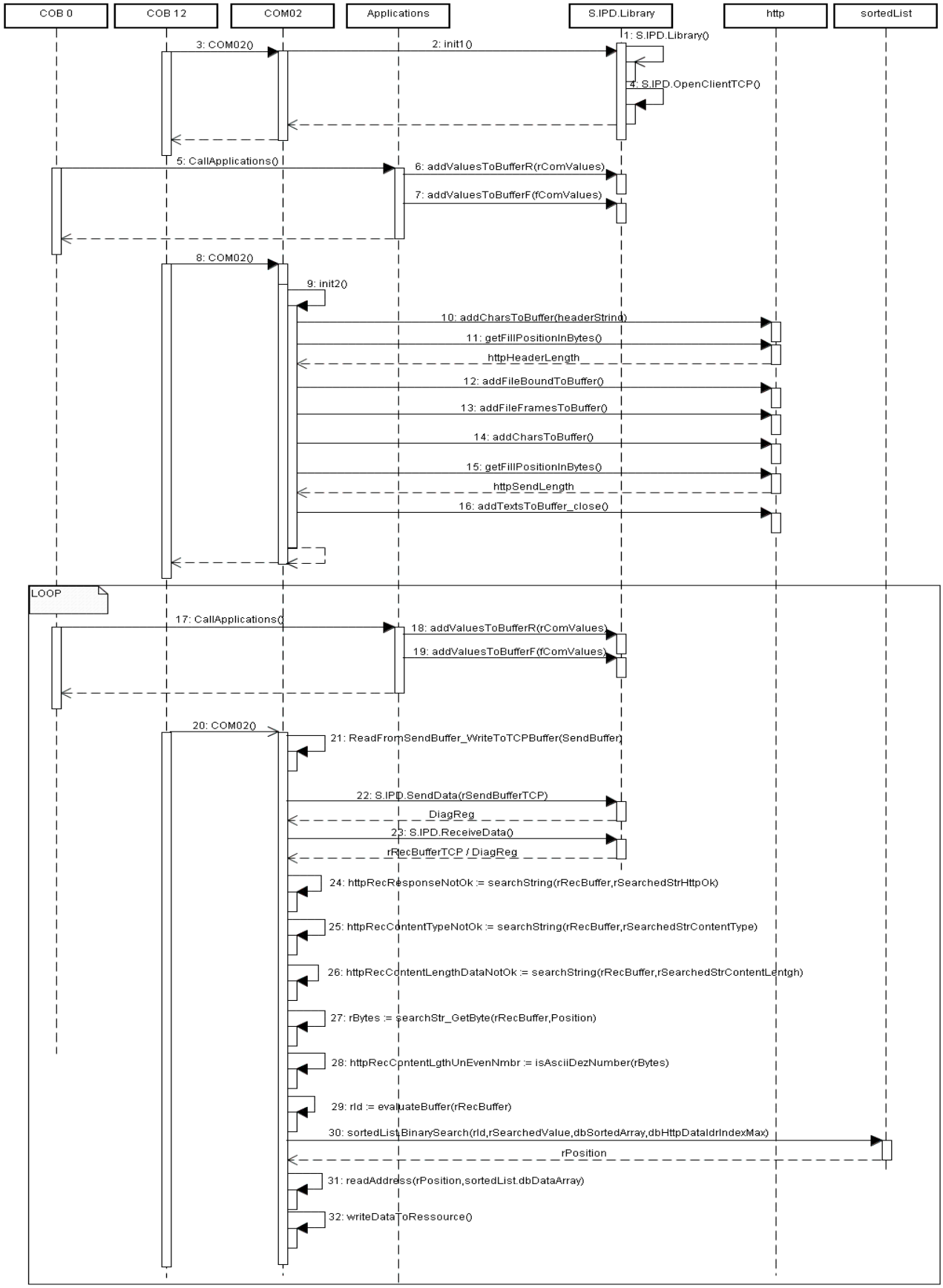


Abbildung 29: Sequenzdiagramm Kommunikation Feldebene

6.2.1 Serviceschnittstelle

Die Serviceebene habe ich mit dem graphikorientierten Leitsystem ProMoSNT realisiert. Dieses läuft auf einem herkömmlichen Windowsrechner. Die Kommunikation mit ProMosNT und dem Feldgerät besteht aus einer Ethernet-UDP- Verbindung. Das Kommunikationsprogramm pollt dabei 2-3 mal in der Sekunde die gewünschten Register und Flags der Steuerung. In der Steuerung selbst muss hierzu keine spezielle Software installiert oder erstellt werden. Die UDP-Anfragen vom Leitsystem werden vom Betriebssystem der Steuerung behandelt. Dieses liefert auf Anfrage, die gewünschten Flags und Register an das Leitsystem. Im Servicefenster können die Fehler und Kommunikationsparameter eingesehen und die Parameter wenn nötig mit einem Mausklick angepasst werden.

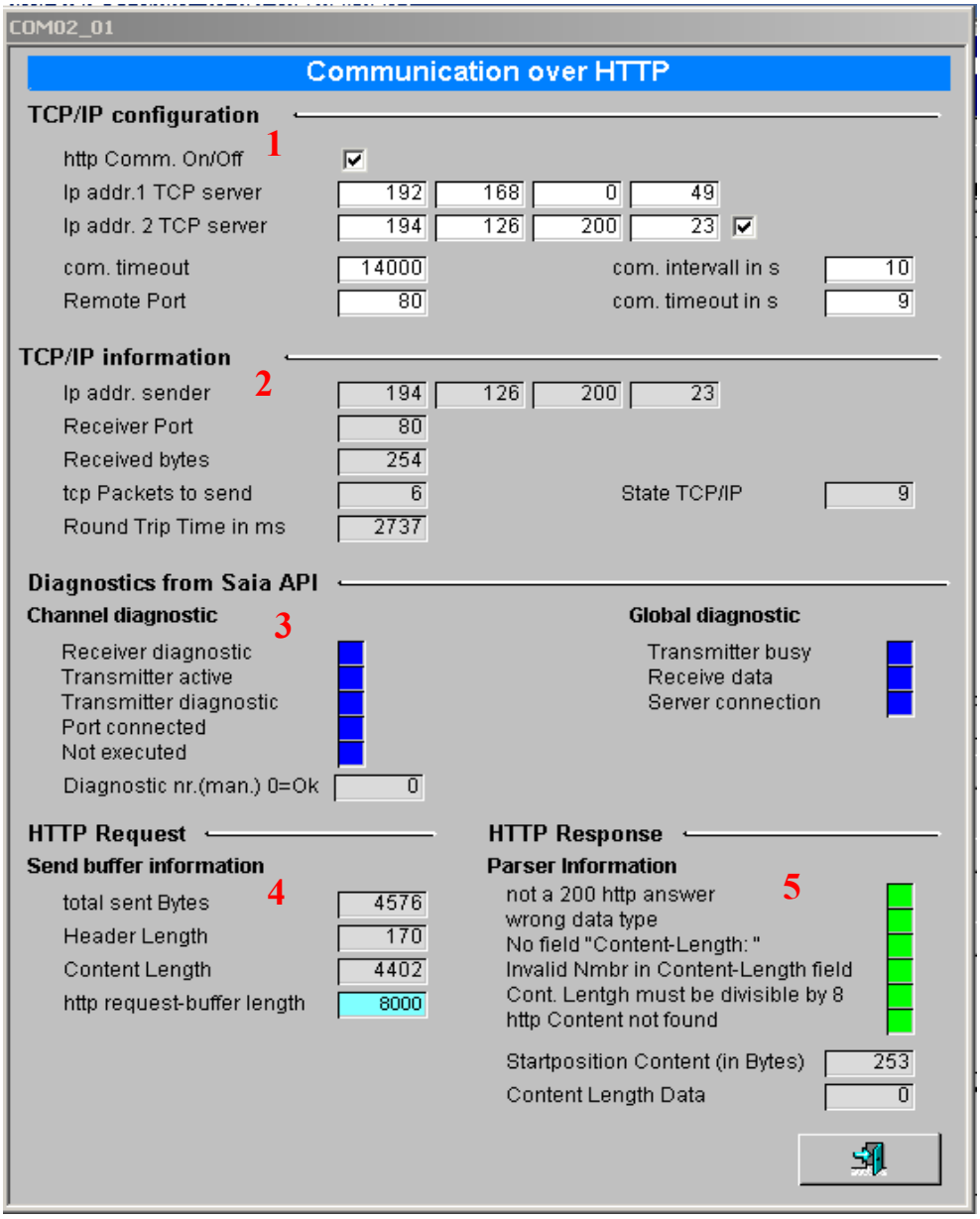


Abbildung 30: Service Panel Kommunikation

1. **TCP/IP Configuration**
   - Mit „**http Comm. On/off**“ kann die Kommunikation aktiviert bzw. deaktiviert werden.
   - Im Feld „**IP addr. 1 TCP Server**“ kann die IP-Adresse des Webservers eingegeben werden
   - Im Feld „**IP addr. 2 TCP Server**“ kann die IP Adresse eines zweiten Webservers angegeben werden. Die Kommunikation zu diesem wird umgestellt wenn das Häkchen dahinter gesetzt ist. Dies ermöglicht ein einfaches Umschalten zwischen einer „Entwicklungsplattform“ und einer Liveplattform.
   - Mit **„com. Timeout**“ kann das TCP/IP-Timeout für das Saia API angegeben werden.
   - Mit „**Remote Port**“ kann der Port des Servers angegebenen werden. Üblicherweise Port 80 für Webserver.
   - „**com intervall**“ gibt an, wie häufig ein Request auf den Server ausgeführt werden soll. Dieser kann auch null sein. Es werden dann im schnellstmöglichen Intervall kontinuierlich Daten gesendet.
   - „**com timeout**“ bricht die Kommunikation nach der angegebenen Zeit ab, egal ob gerade Daten gesendet werden oder ob Daten empfangen werden. Siehe S.36 Abbildung 22: State Event Kommunikation Feldgerät „Communication-Timeout“.

2. TCP/IP Information
   - „**Ip. Addr. Sender**“ zeigt die IP-Adresse von welcher das gesendete IP Datagramm stammt.
   - „**Receiver Port**“ zeigt von welchem Port die gesendeten Daten stammen - „**Received bytes**“ zeigt die Anzahl Daten-bytes, die über TCP empfangen wurden.
   - „**Tcp Packets to send**„ zeigt wie viele TCP Pakete für einen HTTP-Request gesendet werden müssen
   - „**State TCP/IP**“ ist zur Fehlersuche gedacht. Es zeigt in welchem State das Kommunikationsprogramm gerade steckt (Siehe S.36 Abbildung 22: State Event Kommunikation Feldgerät). Die wichtigsten Zustände sind: 6/7 sendTCPI_Packets, 9 receiveTCPI_Packet

3. **Diagnostics from Saia API**
   zeigt die Zustände der IP Schnittstelle. Die Details sind der Saia Dokumentation zu entnehmen.

4. **HTTP-Request**
   In dieser Rubrik können einige Informationen über die an den Server gesendeten Daten eingesehen werden.
   - „**total sent bytes**“ informiert über die gesamte Anzahl Bytes die pro Request an den Server gesendet werden.
   - „**Header Length**“: die Länge des HTT-Headers in Bytes
   - „**Content Length**“: die Länge des HTTP-Contents in Bytes
   - „**http request-buffer length**“ die maximal Länge des Sendbuffers. Das Saia-Betriebssystem lässt keine dynamische allokation von Speicher zu. Daher muss die Länge vom Sendebuffer zum vornherein definiert werden. Er muss grösser definiert werden als „total sent bytes“.

5. **HTTP Response**
   Informiert über Fehlerzustände und Auswertungen des Response vom Web-Server (siehe auch Abbildung 25: Validierung Server Response) Im Fehlerfall

leuchtet ein Feld rot auf. Ist alles i.O. bleibt es grün.
- "**not a 200 answer**": Die Server Antwort beginnt nicht mit „HTTP/1.1 200 OK"
- „**wrong data type**": Kein Feld „Content-Type: "oder es beinhaltet nicht den Datentyp spsx.
- „**no field Content-Length**: ": Die Response beinhaltet kein Feld „Content-Length".
- „**Invalid Nmbr in Content-Length field**": Der Inhalt vom Feld „Content-Length: " ist nicht in numerischer Form.
- „**Cont. Lentgh must be divisible by 8**": Die Länge des Inhalts (Content-Length) ist nicht durch 8 Teilbar. Der Inhalt muss immer aus einem Schlüssel von 4 Byte und einem Werte von 4 Byte Länge bestehen(Abbildung 27 Address lookup t able)
- „**http Content not found**": Der Inhalt ist vom Header durch eine Leerzeile getrennt. Wird keine solche Leerzeile gefunden ist „http Content not found" gesetzt.

7 Datenbank

Das Datenbankschema ist zweiteilig organisiert, alle Tabellen, die das aktuelle Prozessabbild enthalten, haben das Präfix cps_ (Current Process Status) vorangestellt. Daten, die historischer Natur sind, also Aufzeichnungen, haben das Präfix his_ (historisch) und haben jeweils ein Timefeld und ein Datenfeld. Pro Datenbankfeld in den CPS-Tabellen kann eine Loggingfunktion aktiviert werden (siehe auch Kapitel Umsetzung Schnittstelle zu Feldebene S.9). Die Empfangsroutine rec.php erstellt automatisch (falls das entprechende Flag im Konfigurationsfeld recCfg.php aktiviert ist) eine neue historische Tabelle. Die Namensgebung für die historischen Daten ist wie folgt aufgebaut: his_CpsTabname_CpsFeldName_Index. Die Abbildungen 31 und 32 zeigen den Aufbau.

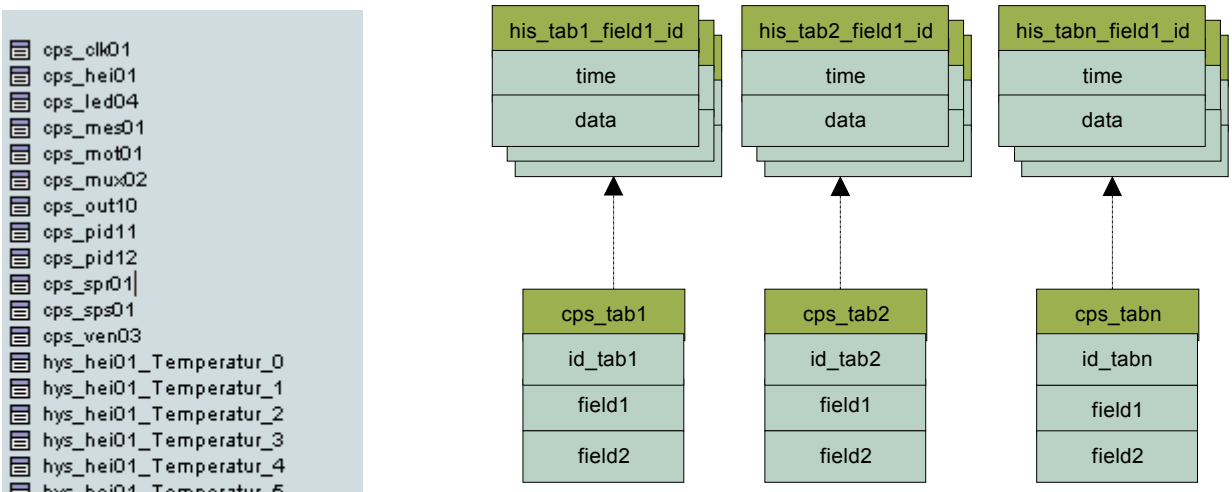


Abbildung 31: ERD Datenbank

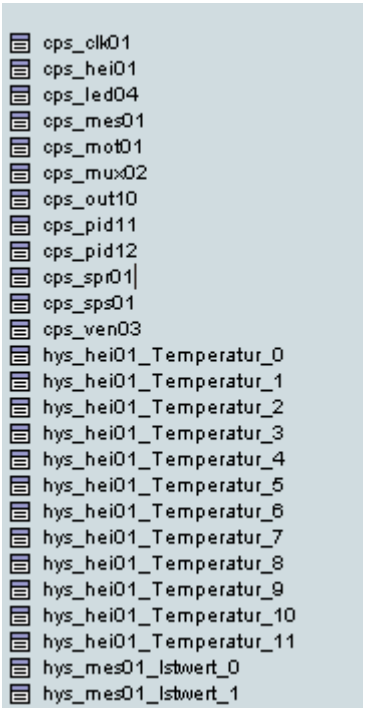


Abbildung 32: ERD umgesetzt

Die historischen Daten sind mit dem Prozessabbild über die Namenskonvention und nicht über Entitäten miteinander verbunden. Ich schätze dies als die sinnvollere und performantere Art ein, um die Daten abzulegen. Die Alternative wäre, die historischen Daten allesamt in eine Tabelle oder eine pro-„cps-Tabelle“ zu schreiben. Die Datensätze müssten dann aber mit Ids dem entsprechenden Feld und dem Datensatz im csp-Teil zugewiesen werden. Historische Daten können aber schnell grosse Datenvolumen annehmen. Müssten die historischen Daten bei jeder Abfrage auseinanderdividiert werden, würde am Schluss die Performance wesentlich leiden.

8 Bedienerinterface

8.1 Design

Die Abbildung 33 gibt eine Übersicht über die Struktur auf dem Webserver. Die ausgezogen Liniene zeigen, welche Files wo includet werden. Die Gepunkteten Linien illustrieren die Verlinkung der Module im Browser untereinander.

Gelb sind die vom Browser direkt abrufbaren Module, rot dargestellt die Konfiguration, grau die allgemein verwendbaren Library-Funktionen, grün die Schnittstelle für die Feldebene. Gut sichtbar der hierarchische Aufbau der Applikation. Die Files in den grauen Felder liegen aus Sicherheitsgründen im nicht- öffentlichen Verzeichnissen des Webservers. Einzig die gelben und grünen Files die als Schnittstelle zu den Clients fungieren, müssen von aussen ansprechbar bzw ausführbar sein. Erkennbar ist im Diagramm die 3-Tier-Architektur. Bestehend Darstellungsschicht (gelb) Geschäftslogik (grau) und der Datenbank (nicht dargestellt). Zählt man die Feldebene dazu, ist es gar eine 5-Tier-Architektur.

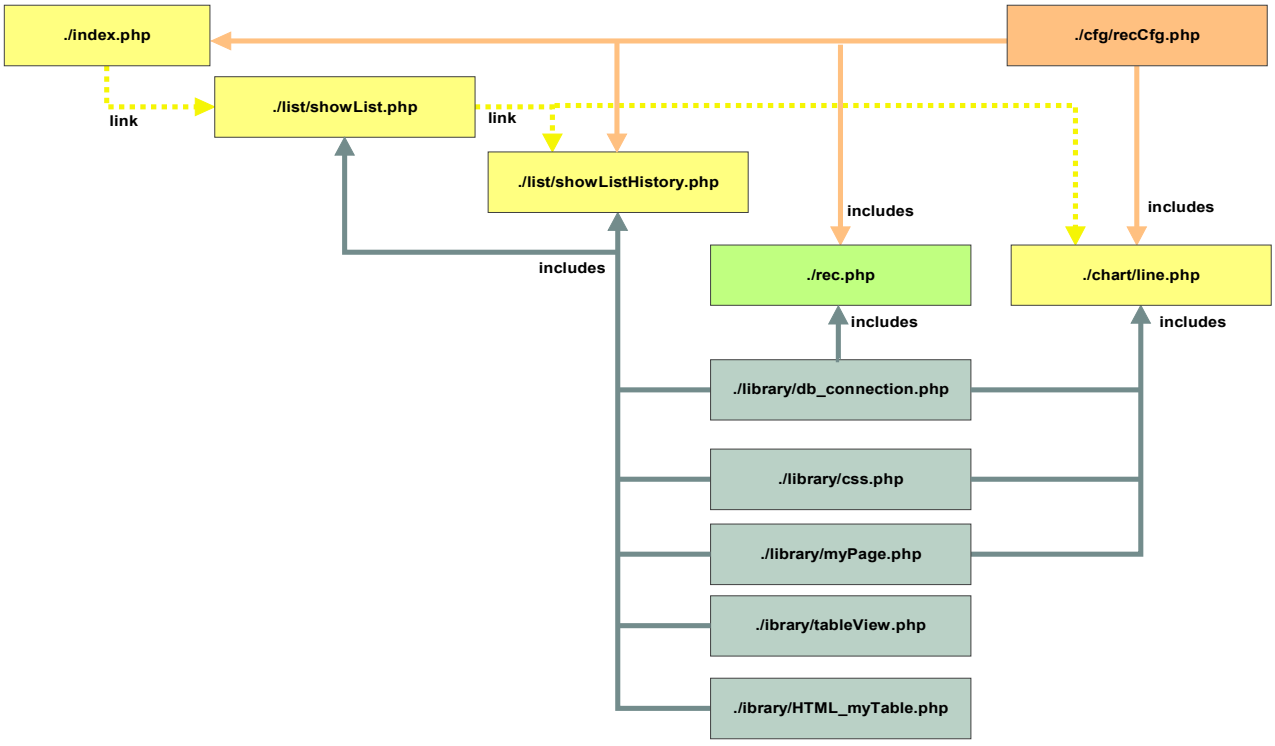


Abbildung 33: Filestruktur Webserver

Damit das Projekt unabhängig von den übergeordneten Pfaden und unabhängig von URLs ist werden durchwegs relative Pfade verwendet.

Die Klassendiagramme der Webanwendung fallen in dieser Anwendung hierarchisch flach aus. D.h. es gibt zur Hauptsache assoziative (Kompositionen) Verbindungen der Klassen. Die Ursache dafür ist die Art der Aufgabenstellungen in Webapplikationen, die häufig aus sequentiellen Abläufen besteht:

→ Request auswerten

→ SQL Statement erstellen

→ Informationen aus Datenbank lesen

→verarbeiten (Geschäftslogik)

→ HTML/CSS/Javascript rendern

Einen grossen Teil der Datenaufbereitung sollte schon mit dem Datenbankschema bzw. mit der Datenbank Abfrage erledigt werden können.

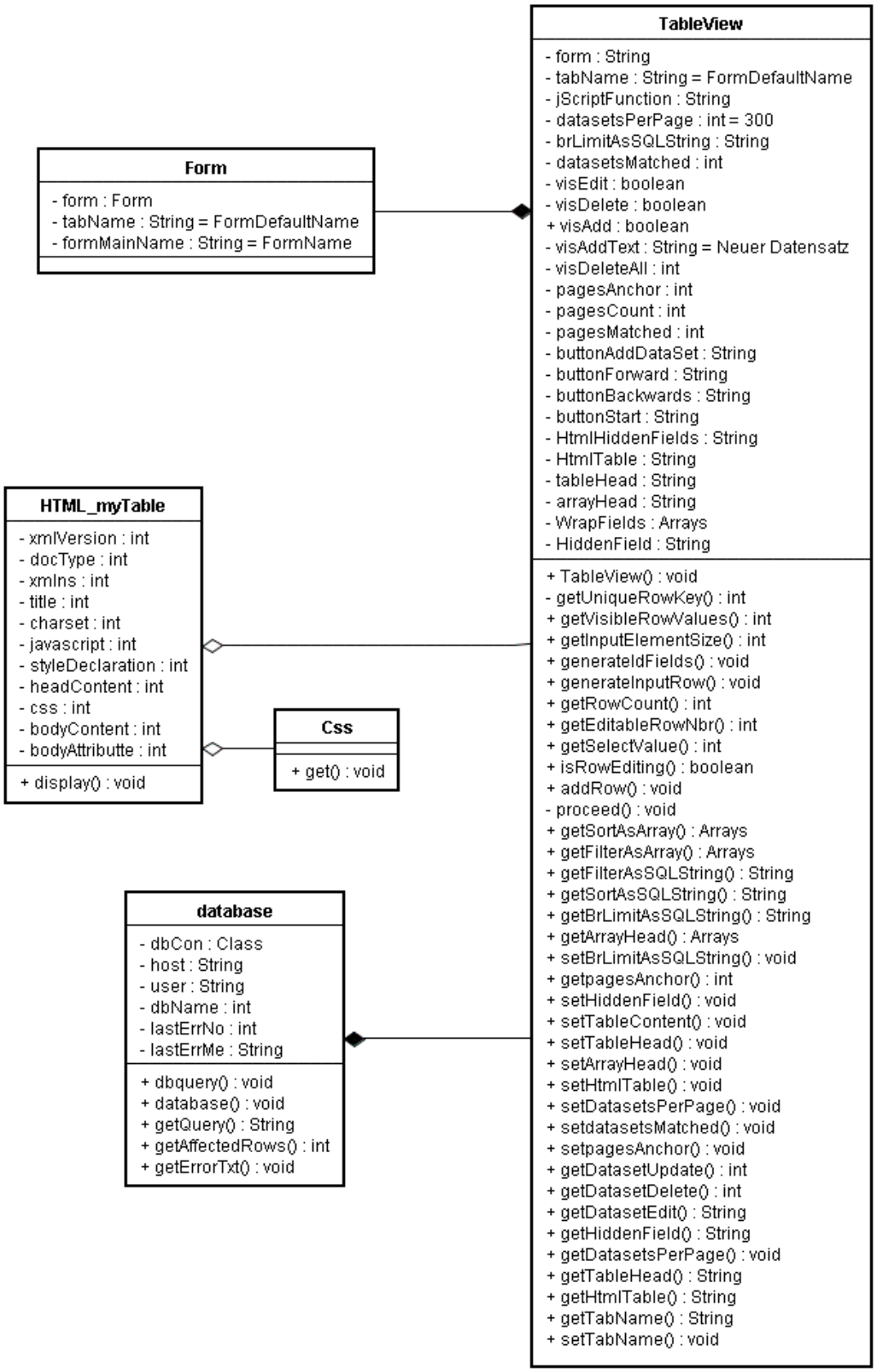
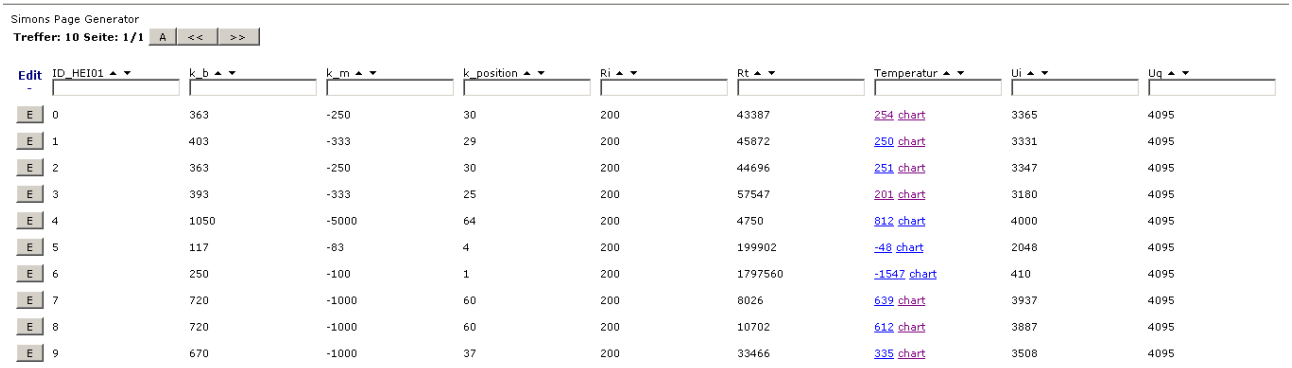

TableView
- form : String
- tabName : String = FormDefaultName
- jScriptFunction : String
- datasetsPerPage : int = 300
- brLimitAsSQLString : String
- datasetsMatched : int
- visEdit : boolean
- visDelete : boolean
+ visAdd : boolean
- visAddText : String = Neuer Datensatz
- visDeleteAll : int
- pagesAnchor : int
- pagesCount : int
- pagesMatched : int
- buttonAddDataSet : String
- buttonForward : String
- buttonBackwards : String
- buttonStart : String
- HtmlHiddenFields : String
- HtmlTable : String
- tableHead : String
- arrayHead : String
- WrapFields : Arrays
- HiddenField : String
+ TableView() : void
- getUniqueRowKey() : int
+ getVisibleRowValues() : int
+ getInputElementSize() : int
+ generateIdFields() : void
+ generateInputRow() : void
+ getRowCount() : int
+ getEditableRowNbr() : int
+ getSelectValue() : int
+ isRowEditing() : boolean
+ addRow() : void
- proceed() : void
+ getSortAsArray() : Arrays
+ getFilterAsArray() : Arrays
+ getFilterAsSQLString() : String
+ getSortAsSQLString() : String
+ getBrLimitAsSQLString() : String
+ getArrayHead() : Arrays
+ setBrLimitAsSQLString() : void
+ getpagesAnchor() : int
+ setHiddenField() : void
+ setTableContent() : void
+ setTableHead() : void
+ setArrayHead() : void
+ setHtmlTable() : void
+ setDatasetsPerPage() : void
+ setdatasetsMatched() : void
+ setpagesAnchor() : void
+ getDatasetUpdate() : int
+ getDatasetDelete() : int
+ getDatasetEdit() : String
+ getHiddenField() : String
+ getDatasetsPerPage() : void
+ getTableHead() : String
+ getHtmlTable() : String
+ getTabName() : String
+ setTabName() : void
Form
- form : Form
- tabName : String = FormDefaultName
- formMainName : String = FormName
HTML_myTable
- xmlVersion : int
- docType : int
- xmlns : int
- title : int
- charset : int
- javascript : int
- styleDeclaration : int
- headContent : int
- css : int
- bodyContent : int
- bodyAttributte : int
+ display() : void
Css
+ get() : void
database
- dbCon : Class
- host : String
- user : String
- dbName : int
- lastErrNo : int
- lastErrMe : String
+ dbquery() : void
+ database() : void
+ getQuery() : String
+ getAffectedRows() : int
+ getErrorTxt() : void

Abbildung 34: Klassendiagramm Bedienerinterface

8.2 Umsetzung

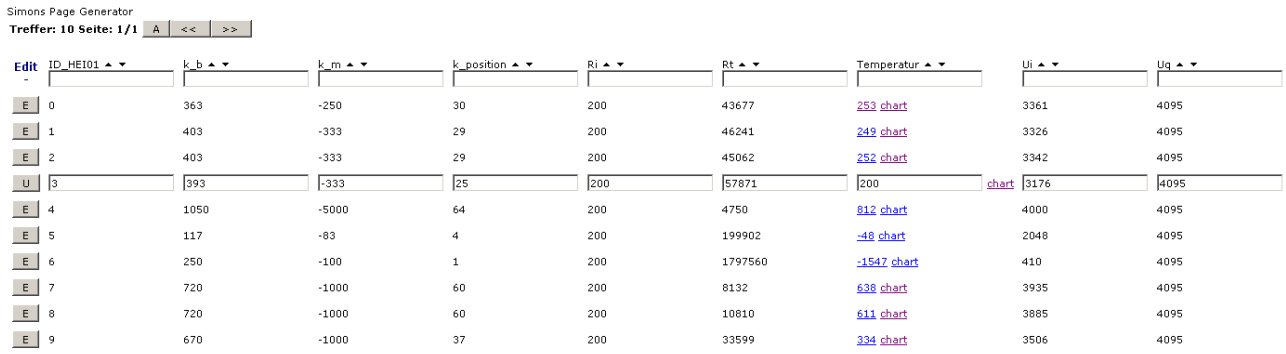
Die Darstellung der Daten ist bewusst einfach und daher übersichtlich gehalten. Als Übersicht dient eine Liste aller abrufbaren Tabellen in der Datenbank. Klickt man eine an, so erscheint eine die Ansicht der Tabelle:


Simons Page Generator
Treffer: 10 Seite: 1/1 A << >>

Edit -	ID_HEI01	k_b	k_m	k_position	Ri	Rt	Temperatur	Ui	Ug
E	0	363	-250	30	200	43387	254 chart	3365	4095
E	1	403	-333	29	200	45872	250 chart	3331	4095
E	2	363	-250	30	200	44696	251 chart	3347	4095
E	3	393	-333	25	200	57547	201 chart	3180	4095
E	4	1050	-5000	64	200	4750	812 chart	4000	4095
E	5	117	-83	4	200	199902	-48 chart	2048	4095
E	6	250	-100	1	200	1797560	-1547 chart	410	4095
E	7	720	-1000	60	200	8026	639 chart	3937	4095
E	8	720	-1000	60	200	10702	612 chart	3887	4095
E	9	670	-1000	37	200	33466	335 chart	3508	4095

Abbildung 35: Ansicht Prozessabbild

Dargestellt wird ein Ausschnitt aus der jeweiligen Datenbank Tabelle, anhängig was im recCfg.php (Kap. 5.2 Umsetzung Schnittstelle zu Feldebene S. 29.) als Kommunikationsfelder angegeben wurde. In der Liste sind Filter und Sortierfunktionen integriert. Mit dem E-Knopf rechts kann kann ein Datensatz editiert werden:


Simons Page Generator
Treffer: 10 Seite: 1/1 A << >>

Edit -	ID_HEI01	k_b	k_m	k_position	Ri	Rt	Temperatur	Ui	Ug
E	0	363	-250	30	200	43677	253 chart	3361	4095
E	1	403	-333	29	200	46241	249 chart	3326	4095
E	2	403	-333	29	200	45062	252 chart	3342	4095
U	3	393	-333	25	200	57871	200 chart	3176	4095
E	4	1050	-5000	64	200	4750	812 chart	4000	4095
E	5	117	-83	4	200	199902	-48 chart	2048	4095
E	6	250	-100	1	200	1797560	-1547 chart	410	4095
E	7	720	-1000	60	200	8132	638 chart	3935	4095
E	8	720	-1000	60	200	10810	611 chart	3885	4095
E	9	670	-1000	37	200	33599	334 chart	3506	4095

Abbildung 36: Editiermodus Prozessabbild

Ein editierter Wert wird mit dem nächsten Response zum Feldgerät übermittelt. Clickt man auf eines der blauen Felder in der Spalte Temperatur, erscheinen die aufgezeichneten Werte des entsprechenden Feldes, wahlweise als Chartaufzeichnug oder als Listenansicht .

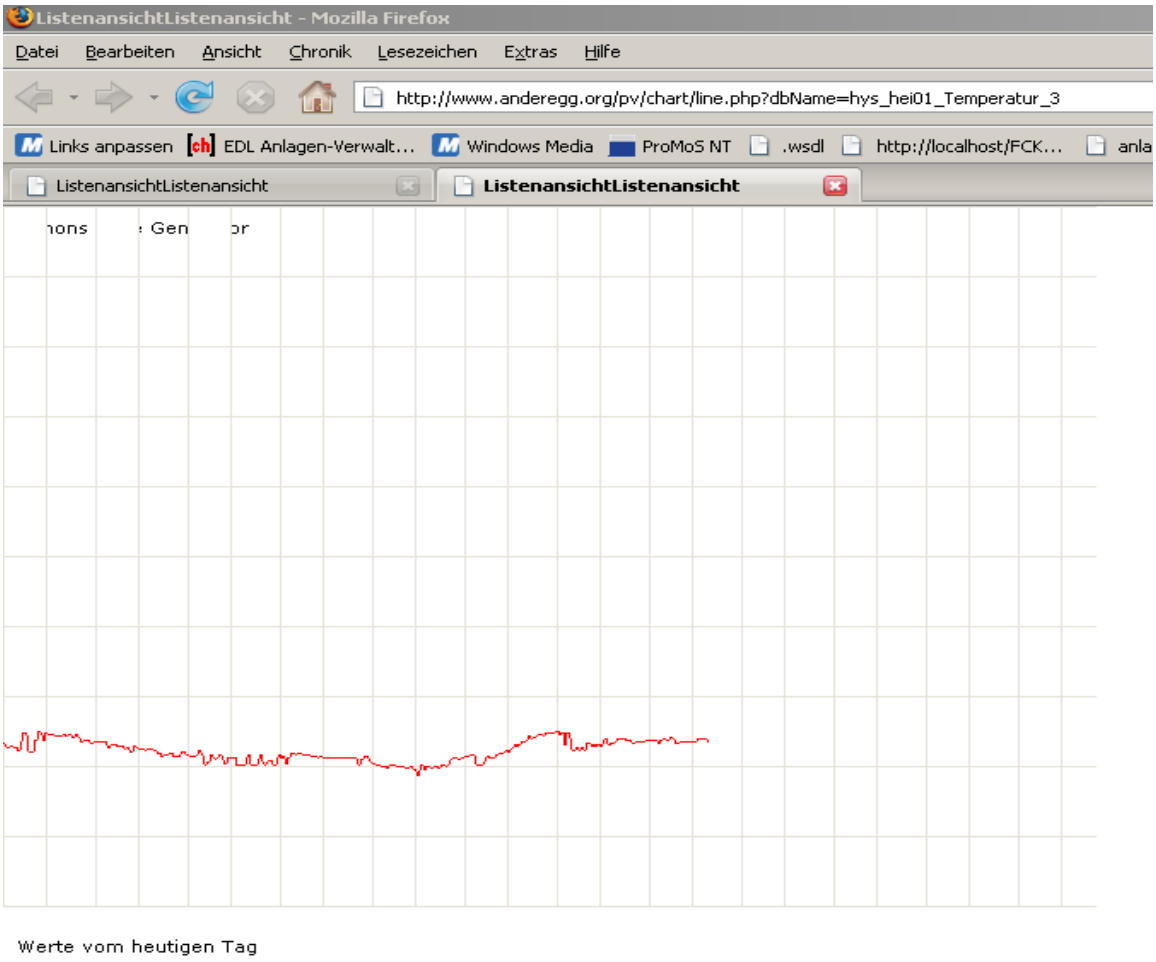


Abbildung 37: Chartaufzeichnung

Die Chartansicht ist noch ausbaufähig. Die Achsen müssten mit Einheiten beschriftet werden, auch könnten eine Blätterfunktion und eine Zoomfunktion integriert werden. Zur Erzeugung der Kurve wurde normales CSS verwendet. Die Ausgabe von CSS und HTML läuft bedeutend schneller ab, als die Generierung eines Bildes im jpg oder ein png Format.

9 Kontrollfunktionen und Tests

9.1 Allgemeines

Das Entwickeln von Software besteht aus planen / designen, implementieren und testen. Entwickeln, debuggen und testen geht Hand in Hand einher. So wird eine Funktion oder Methode schon beim Entstehen, Stück für Stück getestet, anders lassen sich keine funktionierenden Programme erstellen. Es war mir wichtig diese Regeln beim Entwickeln einzuhalten.

9.2 Modultests Feldebene

Die Tests werden mithilfe des Saia-Online-Debuggers getestet. Mit dem Debugger kann direkt auf den Speicher der Steuerung zugegriffen, Breakpoints gesetzt werden etc. Die vollständigen Tests befinden sich im Testdokument. Als Beispiel wird nachfolgend der Test der COM02.SearchString() dokumentiert. Die Testergebnisse der restlichen Tests sind im Testprotokoll zu finden.

Test der Routine COM02.searchString()

Funktionsbeschreibung

Die Routine sucht in einem String (Array von Bytes) nach dem Vorkommen eines zweiten Strings.

Testvariante

Whitebox Test mit dem Debugger

Testbeschreibung

Die Routine bedient sich der Subroutine searchStr_GetByte(), an der mit Index angegeben Position gibt diese immer das angeforderte Byte zurück. Die searchString()-Routine führt jeweils zwei Zeiger mit, einer zeigt auf das aktuelle Byte des Such-Strings und einer zeigt auf das aktuelle Byte des gesuchten Strings. Der Such-String-Zeiger wird kontinuierlich erhöht, während der Gesucht-String-Pointer nur inkrementiert wird, wenn der Inhalt, auf den die beiden Pointer zeigen, identisch ist. Hat der Gesucht-String-Pointer die angegebene Länge erreicht, so gilt der String als gefunden. Der Rückgabewert ist die gefundene Position. Hat der Such-String-Pointer die Länge des Such-Strings erreicht so gilt der String als nicht gefunden. Der Rückgabewert ist -1.

Das Durchsuchen des Strings kann mit dem Debugger nachvollzogen werden:
Such-String:
HTTP/1.1 200 OK
Date: Fri, 15 Aug 2008 10:46:19 GMT
Server: Apache/2.2.0 (Win32) DAV/2 mod_ssl/2.2.0
OpenSSL/0.9.8amod_autoindex_color PHP/5.1.1 X-Powered-By: PHP/5.1.1
Content-Length: 4
Content-Type: application/spsx

Gesuch-String:
Content-Type: application/spsx

Ein Breakpoint wird auf der Zeile gesetzt, auf der die beiden Bytes verglichen werden:

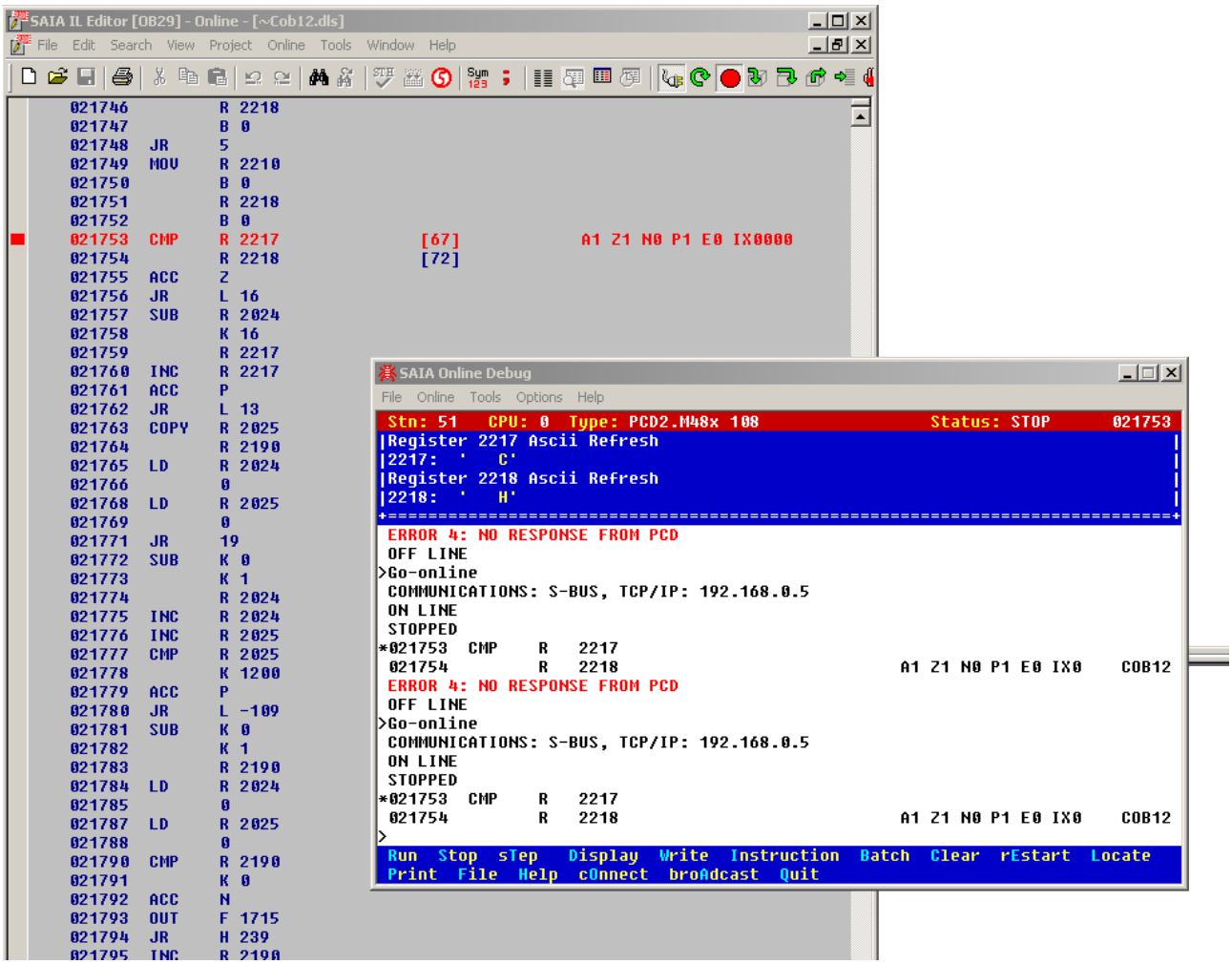
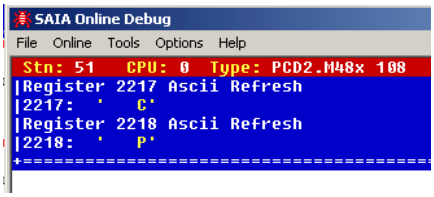
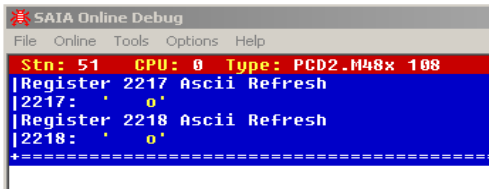


Abbildung 38: Debugging mit dem Saia Programming Tool PG5

Cycle 4, HTTP

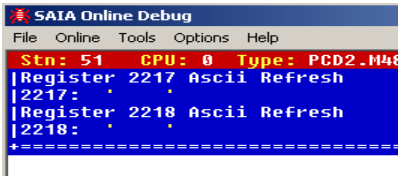

SAIA Online Debug
File Online Tools Options Help
Stn: 51 CPU: 0 Type: PCD2.M48x 108
|Register 2217 Ascii Refresh
|2217: ' C'
|Register 2218 Ascii Refresh
|2218: ' P'

Cycle 175, HTTP/1.1 200 OK Date: Fri, 15 Aug 2008 10:46:19 GMT Server: Apache/2.2.0 (Win32) DAV/2 mod_ssl/2.2.0 OpenSSL/0.9.8a mod_autoindex_color PHP/5.1.1 X-Powered-By: PHP/5.1.1 Content-Length: 4 Content-Type: application/spsx


SAIA Online Debug
File Online Tools Options Help
Stn: 51 CPU: 0 Type: PCD2.M48x 108
|Register 2217 Ascii Refresh
|2217: ' o'
|Register 2218 Ascii Refresh
|2218: ' o'

Bei einem Match wird auch der Pointer auf Gesucht-String inkrementiert.

Cycle 189, Cycle 175, HTTP/1.1 200 OK
Date: Fri, 15 Aug 2008 10:46:19 GMT
Server: Apache/2.2.0 (Win32) DAV/2 mod_ssl/2.2.0 OpenSSL/0.9.8a mod_autoindex_color PHP/5.1.1
X-Powered-By: PHP/5.1.1
Content-Length: 4
Content-Type: application/spsx


SAIA Online Debug
File Online Tools Options Help
Stn: 51 CPU: 0 Type: PCD2.M48
|Register 2217 Ascii Refresh
|2217: ' '
|Register 2218 Ascii Refresh
|2218: ' '

Der Gesucht-Pointer hat die Länge der Gesucht-Strings erreicht, die Schleife wird abgebrochen und die Position der Such-String-Pointers zurückgegeben:

10 Tests

Die korrekte Übermittlung der Daten von der Feldebene in die Datenbank kann mit dem bestehenden Leitsystem verifiziert werden. ProMosNT bietet eine EXCEL- Konnektivität zu ihrer proprietären Datenbank. Nun können alle gewünschten Werte aus der ProMosNT-Datenbank ausgelesen werden. Ebenso könnte mit EXCEL via ODBC auf die MySQL-Datenbank zugegriffen werden oder per copy-paste die Daten vom Web-Frontend, die ebenso in identischer, tabellarischer Form vorliegen, in die Excel-Tabelle eingefügt werden. Die zweite Variante hat den Vorteil, dass das Web-Frontend gleich mitgeprüft wird. Ich habe mich daher für die zweite Variante entschieden (Abbildung 40: Testprinzip). Die restlichen Testtabellen sind im Anhang zu finden. Siehe unten: Tabelle zur Erläuterung der möglichen Testresultate der Funktion MES01(). In der linken Spalte finden sich die Namen der Routinen (Funktion) bzw. deren Funktion. Vertikal sind die Spaltenfelder aufgeführt, rot die mittels ProMosNT direkt aus dem Feldgerät ausgelesenen Werte, gelb die Werte aus der Datenbank, grün das Testresultat.

NAME	id	Eing	Eing_R	Err_SaGroup	Ersatzwert	FiT1_T1	GW_Lo	Istwert	Offset_1	TypFuehler	TypKarte	Umr_X1	Umr_X2	Umr_Y1	Umr_Y2	Err	Ersatz	FBr_Err	FiT1_Aktiv	GW_HE_Err	GW_LE_Err	Quit	Quit_Eing	Umr_Aktiv
Temperatur Vorlauf Öl-Holzkessel	0	64	R.rNull	1	4095	30	2048	3283	??	0	3	0	0	0	0	0	0	0	1	0	0	1	I.120	0
Rücklauf Öl-Holzkessel	1	65	R.rNull	1	3358	20	2048	3273	??	0	3	30	0	0	0	0	0	0	1	0	0	0	I.120	0
Vorlauf Heizung	2	66	R.rNull	1	3358	50	2048	3288	??	0	3	0	0	0	0	0	0	0	1	0	0	0	I.120	0
Aussentemperatur	3	67	R.rNull	1	2867	150	410	3268	??	0	3	0	0	0	0	0	0	0	1	0	0	0	I.120	0
Raumtemperatur	4	68	R.rNull	1	4000	0	0	4000	??	0	3	0	0	0	0	0	1	0	0	0	0	0	I.120	0
Speicher Oben (Tauchf.)	5	69	R.rNull	1	2048	50	0	2048	??	0	3	0	0	0	0	0	1	0	0	0	0	0	I.120	0
Speicher Unten (Tauchf.)	6	70	R.rNull	1	410	0	0	410	??	0	3	0	0	0	0	0	1	0	0	0	0	0	I.120	0
Speicher pos. ca. 1/4	7	71	R.rNull	1	2048	50	2457	3832	??	0	3	0	0	0	0	0	0	0	1	0	0	0	I.120	0
Speicher pos. ca. 1/2	8	80	R.rNull	1	2040	50	2457	3720	??	0	3	0	0	0	0	0	0	0	1	0	0	0	I.120	0
Speicher pos. ca. 3/4	9	81	R.rNull	1	4013	50	2457	3481	??	0	3	0	0	0	0	0	0	0	1	0	0	0	I.120	0
Schaltschrank Temperatur	10	87	R.rNull	1	221	0	50	226	??	2	3	0	0	0	0	0	0	0	0	0	0	0	I.120	0
Referenzspannung der Fühler	11	83	R.rNull	1	4095	0	2048	4095	??	0	3	0	0	0	0	0	0	0	0	0	0	0	I.120	0
Temperatur Vorlauf Öl-Holzkessel	0	64	0	1	4095	30	2048	3283	0	0	3	0	0	0	0	0	0	0	1	0	0	1	0	0
Rücklauf Öl-Holzkessel	1	65	0	1	3358	20	2048	3273	0	0	3	30	0	0	0	0	0	0	1	0	0	0	0	0
Vorlauf Heizung	2	66	0	1	3358	50	2048	3288	0	0	3	0	0	0	0	0	0	0	1	0	0	0	0	0
Aussentemperatur	3	67	0	1	2867	150	410	3267	0	0	3	0	0	0	0	0	0	0	1	0	0	0	0	0
Raumtemperatur	4	68	0	1	4000	0	0	4000	0	0	3	0	0	0	0	0	1	0	0	0	0	0	0	0
Speicher Oben (Tauchf.)	5	69	0	1	2048	50	0	2048	0	0	3	0	0	0	0	0	1	0	0	0	0	0	0	0
Speicher Unten (Tauchf.)	6	70	0	1	410	0	0	410	0	0	3	0	0	0	0	0	1	0	0	0	0	0	0	0
Speicher pos. ca. 1/4	7	71	0	1	2048	50	2457	3832	0	0	3	0	0	0	0	0	0	0	1	0	0	0	0	0
Speicher pos. ca. 1/2	8	80	0	1	2040	50	2457	3720	0	0	3	0	0	0	0	0	0	0	1	0	0	0	0	0
Speicher pos. ca. 3/4	9	81	0	1	4013	50	2457	3481	0	0	3	0	0	0	0	0	0	0	1	0	0	0	0	0
Schaltschrank Temperatur	10	87	0	1	221	0	50	226	0	2	3	0	0	0	0	0	0	0	0	0	0	0	0	0
Referenzspannung der Fühler	11	83	0	1	4095	0	2048	4095	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0
Prüfresultate, id																								
	0	ok	ok	f	ok	ok	ok	ok	ok	f	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	f	ok
	1	ok	ok	f	ok	ok	ok	ok	ok	f	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	f	ok
	2	ok	ok	f	ok	ok	ok	ok	ok	f	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	f	ok
	3	ok	ok	f	ok	ok	ok	ok	f	f	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	f	ok
	4	ok	ok	f	ok	ok	ok	ok	ok	f	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	f	ok
	5	ok	ok	f	ok	ok	ok	ok	ok	f	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	f	ok
	6	ok	ok	f	ok	ok	ok	ok	ok	f	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	f	ok
	7	ok	ok	f	ok	ok	ok	ok	ok	f	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	f	ok
	8	ok	ok	f	ok	ok	ok	ok	ok	f	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	f	ok
	9	ok	ok	f	ok	ok	ok	ok	ok	f	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	f	ok
	10	ok	ok	f	ok	ok	ok	ok	ok	f	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	f	ok
	11	ok	ok	f	ok	ok	ok	ok	ok	f	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	f	ok

Abbildung 39: Integrationstestresultat Funktion MES01()

Als Test werden die Inhalte der gelben und roten Felder mit der EXCEL-Formel „=WENN(C2=C15;"ok";"f")“ verglichen. Alle Felder, welche ein „ok“ enthalten, haben den gleichen Inhalt. Die Felder die ein „f“ beinhalten, stimmen nicht mit einan-

der überein. Es sind tatsächlich einige Differenzen auszumachen. In den Spalten Eing_R und Quit_Eing ist eine Variable aufgeführt. In der MySQL-Datenbank steht jedoch der Inhalt der Variable. Desweiteren stimmen einige Ist-Werte nicht überein; dies liegt an der zeitlich verschobenen Auslesung der Daten. Prüft man die Werte genauer, so sieht man, dass sie nahe beieinander liegen und daher plausibel sind.

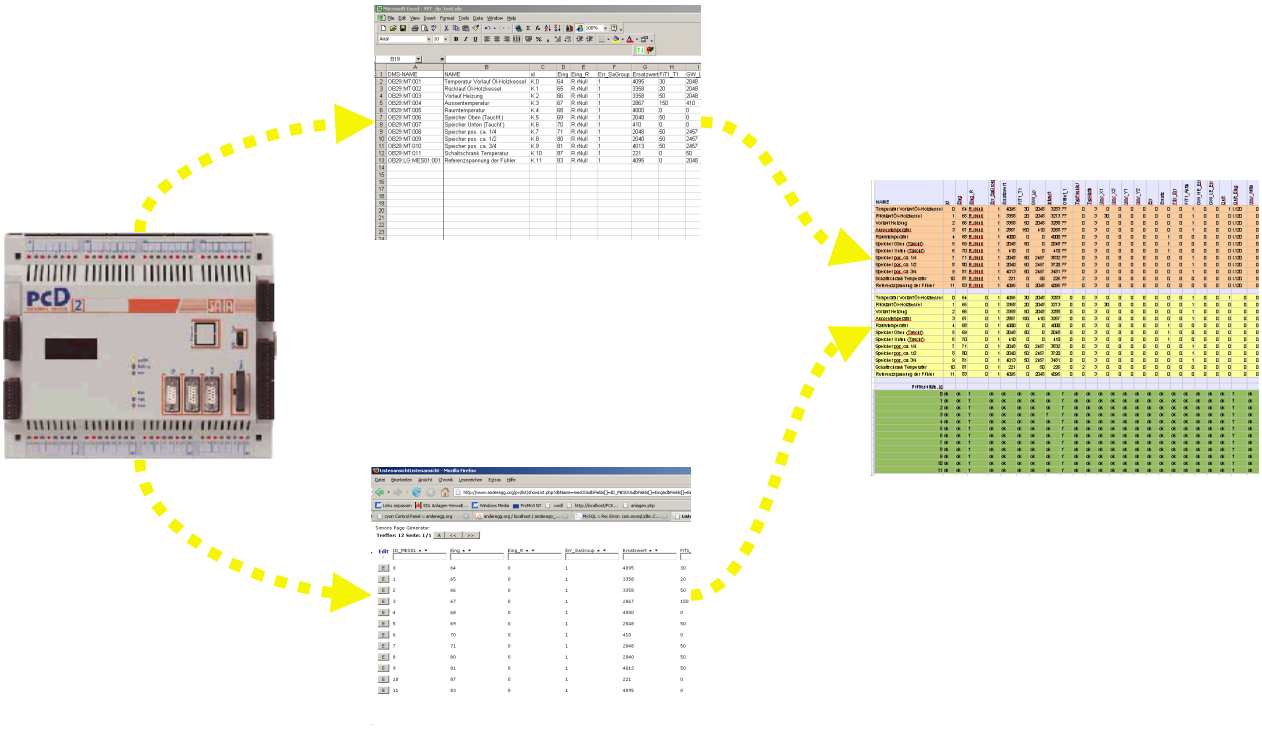


Abbildung 40: Testprinzip

Wie den anderen Protokollen entnommen werden kann, ist der Test bis auf einen Punkt zur Zufriedenheit ausgefallen:

Bei der Funktion SPR01() wurde dem Makro http.addValuesToBufferR() irrtümlicherweise ein Flag als Parameter anstelle eines Registers mitgegeben. In diesem Falle müsste eigentlich der Saia-Compiler reklamieren, weil in der Routine eine copy-Operation auf ein Flag ausgeführt wird. Dies tut er aber überraschenderweise nicht. Es wird daher zu einem Laufzeitfehler im Feldgerät kommen. Die Steuerung bringt dies offenbar nicht aus dem tritt, sie läuft weiter und ignoriert die fehlerhafte Instruktion. Aufgrund der ignorierten Instruktion wird allerdings ein falscher Wert an den Webserver übermittelt. Ich habe nun das http.addValuesToBufferR() Makro dahingehend angepasst dass bei einem Flag als Parameter, ein Warning beim Compilieren generiert wird. Ignoriert man das Warning, wird das Flag trotzdem korrekt, aber mit 31 Bits, an Overhead übermittelt.

11 Das HTTP-Protokoll

Das Http-Protokoll wird verwendet um Daten über ein Netzwerk zu transportieren. Das Http-Protokoll entstand zusammen mit HTML am Cern in Genf, aus dem Bedürfnis der Wissenschaftler heraus, Daten auf elektronischem Weg untereinander auszutauschen. Das HTTP-Protokoll selbst ist zustandslos und basiert auf dem TCP/IP-Protokoll. Da TCP/IP eine fehlerfreie Übertragungsart ist (mit checksum), kann diesbezüglich auch das HTTP-Protokoll als fehlerfreie Übertragungsart angesehen werden. Um Binärdaten zu senden, sieht das HTTP-Protokoll das Multipart form data vor. Mehr dazu lässt sich im RFC 1867 und RFC 1521 unter iana.org (Internet Corporation for Assigned Names and Numbers) nachlesen.

12 Manipulationssicherheit

12.1 Allgemeines

Beim jetzigen Prototypen wurde die Sicherheit nicht prioritär behandelt. Auf den ersten Blick ist man versucht zu sagen, die Daten von Geräteüberwachungen etc. böten kein lohnenswertes Ziel für Hacker. Die überwachten Geräte sind aber nicht einfach nur Selbstzweck, sondern haben bestimmte Funktionen und Aufgaben zu erfüllen. Sie steuern Prozesse in Fabrikationen oder ganze Gebäudeautomationen. In vielen Fällen ist heute in einem modernen Gebäude vom Zutrittsystem über Licht, Jalousien, Heizung und Klimaanlage alles miteinander vernetzt. Wäre es jemandem möglich, Kontrolle via Fernwartung über ein oder mehrere Geräte zu erlangen, so könnte dieser gleichwohl Schaden anrichten oder an sensible Daten wie An- oder Abwesenheitszeiten von Personen etc. gelangen. Für zwielichtige Personen wäre es beispielsweise interessant zu wissen, wann der Chef nicht im Büro oder wann der Villenbesitzer ausser Haus ist usw. Bei heutigen Systemen mit Präsenzschalter, welche natürlich den Komfort erhöhen, lassen sich schon heute Räume überwachen und Zutritte aufzeichnen. Heute sind solche Systeme vielfach proprietär und die Daten bleiben zumeist Inhouse. Somit sind mögliche Hackerattacken auf einen kleinen Personenkreis begrenzt, welche erstens Zutritt zum Gebäude haben, zweitens Zugriff zum System und drittens über das nötige Spezialwissen mit den nötigen Gerätschaften verfügen. Deshalb gewinnt die Problematik eine neue Dimension wenn solche Daten nun auf einen Webserver geschrieben werden, wo die Hackerattacken von überall her gestartet werden können und somit eine grössere Anzahl Personen über das nötige Spezialwissen für Attacken auf einen Webserver verfügen. Dennoch bin ich überzeugt, dass sich eine zufriedenstellende Sicherheit erreichen liesse. Zum einen durch die heute üblichen Verschlüsselungs- und Authentifizierungsverfahren, zum anderen verhindert eine konservative Datenaufschaltung auch das Schadenpotenzial.

Die Datensicherheit soll hier nicht eingehend behandelt werden, wäre es doch für sich alleine ein buchfüllendes Thema.

12.2 Sicherheit Kommunikation Feldebene – Webserver

Mit Sicherheit ist hier die Abhör- und Manipulationssicherheit der Kommunikation zwischen Feldebene und Webserver gemeint. Die fehlerfreie Übertragungssicherheit

wird durch das TCP-Protokoll sichergestellt. Im kryptographischen Sinn wird Sicherheit wie folgt definiert:

- **Confidentiality**: Die Vertraulichkeit stellt sicher, dass die Nachricht niemand anderes lesen kann als der vorgesehene Empfänger.
- **Authentication**: Die Authentikation stellt sicher, dass die empfangene Nachricht auch von dem Sender stammt, als der er sich ausgibt.
- **Integrity**: Mit der Integrität kann überprüft werden, ob jemand die gesendeten Daten manipuliert hat.
- **Nonreputation**: (Die Reputation) Der Empfänger kann nicht verleugnen, die Nachricht erhalten zu haben.

In diesem Projekt konnte keinem dieser Punkte Beachtung geschenkt werden. Standard HTTP geht unverschlüsselt übers Netz. D.h. jeder, der sich an geeigneter Stelle einklinkt, kann mitlesen.

Ebenfalls kann auf dem Webserver nicht garantiert werden, dass die Daten auch wirklich vom erwarteten Feldgerät stammen. Es könnte sich auch eine Drittpartie als das erwartete Feldgerät ausgeben.

Die Daten sind nicht manipulationssicher. „The man in the middle“ könnte die Daten verändern, gelingt es ihm, sich zwischen die Kommunikationspartner zu schalten.

Ebenfalls könnte der Webserver-Betreiber leugnen, Datenpakete erhalten zu haben. Dieser Punkt ist wohl nicht der wichtigste der vier Sicherheitsregeln, könnte aber in manchen Anwendungen praktisch sein.

Möchte man die ersten drei Punkte erfüllt haben, so müsste man beim jetzigen Projektstand ein VPN–Tunnel zwischen den beiden Kommunikationspartnern einrichten.

12.3 Sicherheit Kommunikation Webserver - Browser

Die Sicherheitsbetrachtungen vom vorangehenden Kapitel gelten auch hier. Jedoch liesse sich die Sicherheit der Kommunikation zwischen Webserver und Browser einfacher und ohne zusätzliche Hardware massgeblich steigern, zum Beispiel mit einer Verschlüsselung durch HTTPS. Des weiteren müsste ein Authentifikationszertifikat (zum Beispiel von Verisign) gekauft werden. Die Sicherheit liesse sich weiter durch Streichlisten verbessern, wie sie heute beim E-Banking zur Anwendung kommen.

12.4 Sicherheit Datenverarbeitungsebene

Es gibt Einiges, das auf der Datenverabeitungsebene, sprich Webserver und Datenbank, beachtet werden sollte. Eines davon ist SQL injection. Der Angreifer versucht via Webserver beispielsweise mit der Post- oder Get-Methode eigene SQL- Befehle in die bestehende Abfrage des Scripts zu schmuggeln. Unterbinden kann man dies, indem die SQL-Abfragen, die ja zumeist dynamisch, d.h. vom serverseitigen Skript generiert werden, die Variablen konsequent quotet und deren Inhalt vor der Verarbeitung prüft.

Dem Webserver-Client der Datenbank sollten nur die Rechte zugewiesen werden, welche dieser für seine Arbeit benötigt. Wenn er beispielsweise nur lesen muss, so werden ihm keine Rechte für „CREATE“ gegeben.

Die so genannten globalen Variablen sollten deaktiviert werden: PHP kann automatisch Variablen für Daten erzeugen, die per GET oder POST an den Webserver gesendet werden.

Öffentlich zugänglich gemacht werden sollen nur die Verzeichnisse und Dateien auf dem Webserver, welche von aussen angesprochen werden müssen. Verzeichnis-Listings sollten keine ausgegeben werden.

12.5 Sicherheit Feldebene

Da die Feldebene als TCP-Client arbeitet, ist sie gegenüber Manipulationen weniger bzw. gar nicht gefährdet, vorausgesetzt der Response wird ignoriert. Wird der Response ausgewertet, so könnte dieser möglicherweise manipuliert worden sein.

Nicht zu vergessen ist, dass andere Feldgeräte, die einen Mini-Webserver integriert haben, angreifbarer sind, da man direkt mit ihnen interagieren und so gezielt nach Schwächen oder Einbruchsmöglichkeiten suchen kann. Dies ist nicht möglich, wenn das Feldgerät als HTTP-Client arbeitet und der Response ignoriert wird.

13 Datenbasis

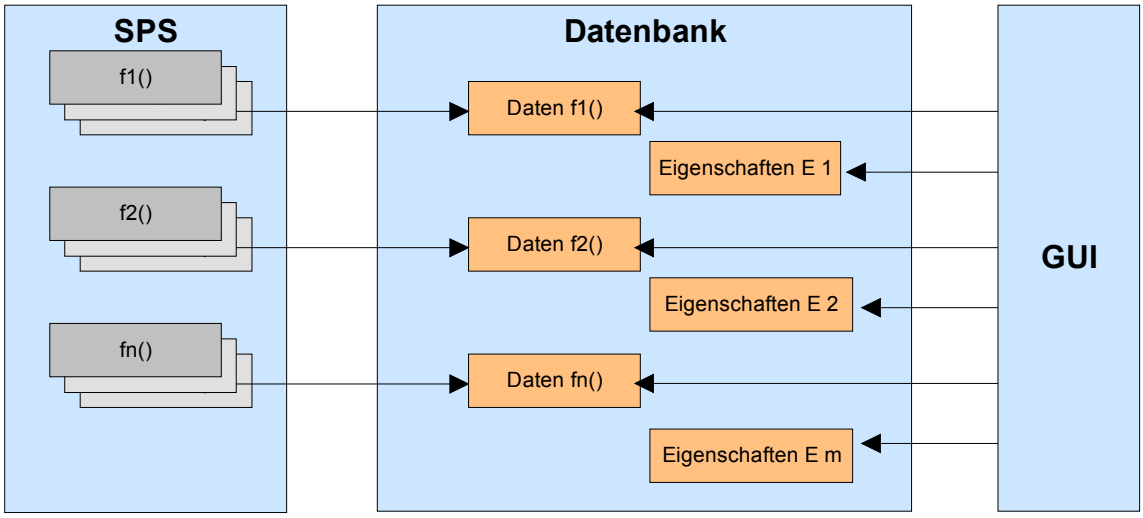


Abbildung 41: Datenflüsse und Erzeuger

Die Daten sind auf der Feldebene bereits nach Kategorien oder Gleichartigkeit gegliedert sein. Um auf das Beispiel der Heizung zurückzukommen, wären Pumpen eine Kategorie, Ventile und Messfühler weitere Kategorien. Bei den allermeisten Maschinen oder Automaten können solche Funktionsgruppen gebildet werden. Ein Softwareentwickler wird daraus Funktionseinheiten bilden. In der Software unseres Heizungsbeispiels sind die Anlagenelemente in Funktionen oder, im Saia-Jargon, in Funktionsblöcke gekapselt.

Diese Daten werden via das entwickelte Softwaremodul auf den Webserver kommuniziert.

Die Feldebene besteht aus den folgenden wesentlichen Softwareelementen, welche Informationsgehalt von Interesse aufweisen:

Messfühler
Temperatur Vorlauf Heizung
Öl-Holzkessel Rücklauf
Öl-Holzkessel Vorlauf
Aussentemperatur
Raumtemperatur
Speicher Oben (Tauchf.)
Speicher Unten (Tauchf.)
Speicher pos. ¼
Speicher pos. ½
Speicher pos. ¾

Ventile
Mischventil Ladung
Speicher Mischventil
Heizung

Regler
Heizungsregler
Kessel-Hochhaltung

Pumpen
Ladepumpe Speicher
Umwälzpumpe Heizung

Schaltuhren
Anwesenheit (Heizung)
Nachttarif (Ölkessel einschalten)

Speicherbewirtschaftung
Speicherladungs-Steuerung

Bedientableaux
Bedientaster LED–Anzeige

13.1 Hardwareschnittstellen

Die Hardwareinstallation der Feldebene besteht aus einer Saia-SPS mit digitalen und analogen IOs. Diese stellen die elektrischen Verbindungen zur Anlage dar. Die Signale der IOs steuern mittels Koppelrelais und Schützen die Anlage. In unserem Fall besteht die Anlage aus einer kombinierten Öl/Holz-Heizungsanlage, welche exemplarisch die Möglichkeiten des realisierten Systems zeigen soll.

Abbildung unten: Ansicht der bestehenden Schaltschrank-Installation.

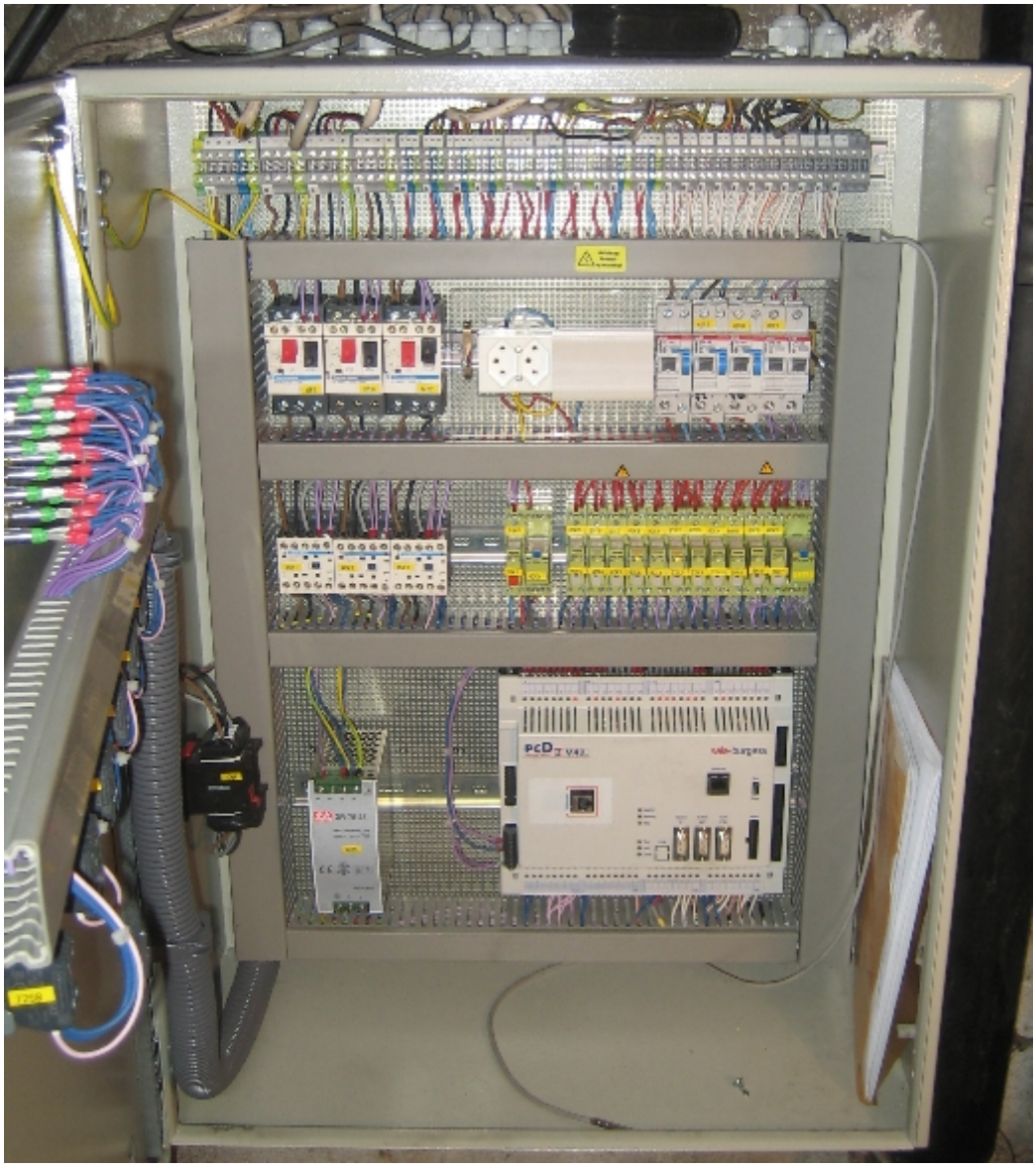


Abbildung 42: Installation Feldebene

14 Resumée

Software ist vergleichbar mit einem Uhrwerk, Zahn für Zahn müssen die Zahnräder ineinander greifen, um am Schluss ein funktionierendes Ganzes zu geben. Klemmt eines der unzähligen Zahnrädchen im Uhrwerk, so steht es still. Wie nun das klemmende Rädchen finden? Zu einem funktionierenden Software-Uhrwerk gehört auch eine einwandfrei funktionierende Infrastruktur. Eine zentrale Funktion hat in diesem Projekt das Netzwerk. So traten eine gewisse Zeit immer Ausfälle der Verbindung des Feldgerätes zum Webserver auf. Die Verbindung war nur zeitweilig unterbrochen, am Anfang ca. alle 1-2 Wochen für ein paar Stunden. Dann wurden die Ausfälle zusehends häufiger. Ich fragte mich: Was könnte die Ursache sein? Ein fehlerhaft funktionierendes ADSL-Modem? Oder liegt der Fehler im lokalen Netzwerk beim Feldgerät? Oder gar am Feldgerät selbst? Nach längerem Suchen fand ich heraus, dass am hausinternen Telefonnetz ein Telefon ohne ADSL-Mikrofilter betrieben wurde, welches die Störungen verursachte. Ein ohne Mikrofilter betriebenes Telefon kann die Internetverbindung stören, tut es aber oft nicht oder nur sporadisch.

Eine weitere Schwierigkeit war die Saia-Steuerung, die, wenn man mehr als 5kB übermittelt, jeweils das TCP-RST-Flag sendet und damit die Kommunikation zu früh beendet. Auch der Hersteller konnte mir in diesem Punkt nicht weiterhelfen. Trotz Austausch von Wireshark-Aufzeichnungen und Nachbau der Anlage (mit meinem Programm) beim Hersteller gelang es nicht, die Fehlerquelle zu orten. Dieses Problem wird später noch gelöst werden müssen. Im Moment können aus diesem Grund nicht die gesamten Daten übermittelt werden.

Ebenso hab ich festgestellt dass sich der Ausgabe-Buffer von php manchmal etwas unberechenbar verhalten kann, wenn im gleichen Skript, File-Operationen (Fehler und Debug-Daten loggen) gemacht werden. So sendeten die Funktionen

```
header('Content-type: application/spsx');
header('Content-Length: '.strlen($uplFileToDB->outputString));
echo $uplFileToDB->outputString;
```

nach dem ausgeben der Header-Daten jeweils zwei CR,LF und dann fälschlicherweise, folgte noch ein LF. Dies hat zur Folge dass das LF als Modul-Id in der SPS interpretiert wird, und damit eine falsche Id ausgwertet wird. Dies kann natürlich gefährlich sein. Da bekannt ist welches Modul die Id 10 besitzt(LN = Ascii 10), weiss man in etwa in welchen Bereich des Speichers die falschen Daten hineingeschrieben werden. Diesem Problem muss man sicher weitere Aufmerksamkeit schenken.

Liegt das fertige Uhrwerk schlussendlich vor, blickt der Uhrmacher trotz all den Mühen mit einem gewissem Stolz auf das tickende Erzeugnis und überlegt, wozu seine Entwicklung auch noch genutzt werden könnte.

15 Ausblick

Mit der vorliegenden Arbeit ist die Basis geschaffen für Fernwartungstechnik für Feldgeräte mi Datenaustausch über einen Webserver. Ein wichtiger Punkt, das Protokoll (Übertragungsart) ist definiert und eine Referenzanlage implementiert. Die Feldebene (für Saia-Geräte) ist relativ weit gediehen. Ein Schönheitsfehler ist klar die fehlende DNS-Auflösung. Diese könnte in einem weiteren Schritt implementiert werden. Ebenso könnte in Sachen Verschlüsselung noch ein weiteres Projekt gestartet werden. Eine Implementierung für andere Feldgeräte wäre ebenso denkbar. Die Datenverabeitungsebene könnte aber auch als M2M-Plattform dienen. D.h. Geräte der Firma A könnten via Webserver mit Geräten der Firma B Daten austauschen. Dies in einem LAN oder dezentral weltweit verteilt.

Auf der Datenverabeitungsebene liessen sich diverse weitere Datenauswertungstools realisieren. Für viele Industrieanwendungen wäre dies nicht einmal nötig, der bestehende Funktionsumfang würde genügen, da es vielfach nur darum geht, Prozessdaten einem anderem System zur Verfügung zu stellen. Dieses verarbeitet die Daten dann weiter. Soll ein Man-Machine-Interface geschaffen werden, gibt es noch viel zu tun. Datenumrechnungen in die eigentlichen physikalischen Grössen ist nur ein Stichwort. Ebenso liesse sich die Datenaufzeichnung beliebig verfeinern, damit sie beispielsweise per Mausklick aktiviert werden kann oder nur Werte aufgezeichnet werden, die ausserhalb einer gewissen Toleranz liegen. Eine weitere Idee wäre, eine graphischen Oberfläche zu realisieren.

Da die Datenorganisation und das Übertagunsprotokoll per HTTP upload definiert ist, könnte diese Lösung serverseitig auch auf anderen Plattformen implementiert werden.

16 Quellenangaben

Literatur:

Name	Autor	Verlag
Cryptographie Decrypt-pet	H:X. Mel; Doris Baker	ADDISON WESSLEY
Objektorientiertes Analyse und Design	Rolf Wenger	Berner Fachhochschule
Datenkommunikation und Rechnernetze	Rolf Lanz	Berner Fachhochschule
Relationale Datenbanken	Arno Schmidhauser	Berner Fachhochschule
Algorithmen und Datenstrukturen	Dr. Beatrice Amrhein	Berner Fachhochschule
Handbuch SAIA	-	Saia Burgess AG
Computer&Automation	-	WEKA FACHMEDIEN GmbH

Online Quellen:

Adresse
www.php.net
www.mysql.com
www.sbc-support.ch

17 Abbildungsverzeichnis

Abbildung 1: Hierarchie Modell nach ISA-95.....5
Abbildung 2: Lösungsansatz Kunde 1.....6
Abbildung 3: Lösungsansatz Kunde 2 7
Abbildung 4: Übersicht Beispiel-Implementation.....7
Abbildung 5: Übersicht use-case 8
Abbildung 6: Abgrenzung Feldebene.....10
Abbildung 7: PCD2.M480.....12
Abbildung 8: Blockschema PCD2.M480 (Benutzerhandbuch Saia).....12
Abbildung 9: Programmierumgebung Saia.....13
Abbildung 10: Prgrammbeispiel in Bloctec.....15
Abbildung 11: Kommunikationsstruktur mit Zentraler Datenverarbeitung.....18
Abbildung 12: Datentransfer-Nutzdaten.....20
Abbildung 13: Datenfluss Feldebene-Datenverarbeitung.....25
Abbildung 14: Inhalt eines hochgeladenen Files.....26
Abbildung 15: Organisation der Daten pro Request.....27
Abbildung 16: Datenfluss Feldgerät.....29
Abbildung 17: Interpretation eines hochgeladenen Files.....30
Abbildung 18: Zustandsdiagramm File Upload.....31
Abbildung 19: Klassendiagramm Datenverarbeitung.....33
Abbildung 20: Daten senden 34
Abbildung 21: Sequenzdiagramm Gesamtablauf.....35
Abbildung 22: State Event Kommunikation Feldgerät.....36
Abbildung 23: Aufbau Sendbuffer.....37
Abbildung 24: Empfangen, HTTP-Response des Webservers.....37
Abbildung 25: Validierung Server Response.....38
Abbildung 26: Datenorganisation im Empfangsbuffer.....38
Abbildung 27: Address lookup t able.....39
Abbildung 28: Klassendiagramm Kommunikation Feldebene.....43
Abbildung 29: Sequenzdiagramm Kommunikation Feldebene.....48
Abbildung 30: Service Panel Kommunikation.....49
Abbildung 31: ERD Datenbank.....52
Abbildung 32: ERD umgesetzt.....52
Abbildung 33: Filestruktur Webserver.....53
Abbildung 34: Klassendiagramm Bedienerinterface.....55
Abbildung 35: Ansicht Prozessabbild.....56
Abbildung 36: Editiermodus Prozessabbild.....56
Abbildung 37: Chartaufzeichnung.....57
Abbildung 38: Debugging mit dem Saia Programming Tool PG5.....59
Abbildung 39: Integrationstestresultat Funktion MES01().....61
Abbildung 40: Testprinzip.....62
Abbildung 41: Datenflüsse und Erzeuger.....66
Abbildung 42: Installation Feldebene.....68